Déclaration de cTxt

```
// v3.35 Tanguy PRUVOT - 01 Oct 2010 - Windev 12-15
// ============================================================
// Description :
// ------------------------------------------------------------
// Transforme une chaine délimitée en tableau à deux dimensions
// Comporte les interfaces d'E/S : >TXT>, >CSV>, >INI>, >Mem>, >Reg, Rep>, >HTML, cTxt>

// Utilisation rapide :

// tTxt est un cTxt(Chaine)
// POUR L=1 a tTxt:nbLines
//   Trace(tTxt:Cell[L,1])
// FIN

// PS: la fonction tTxt:TraceTxt() permet d'afficher le contenu de l'instance dans une trace formatée
pour la visualisation


// ou

// tTxt est un cTxt("",sSepLignePerso,sSepColonnePerso)

// ...

// tTxt:Parse(Chaine)
// POUR L=1 a tTxt:nbLines
//   POUR C=1 a tTxt:nbCols
//      Trace(tTxt:Cell[L,C])
//   FIN
// FIN

// ============================================================
// Historique :
//  légende (+ ajout, * bugfix, o optimisation compatible, x modif ou suppression non retro compat., d
notes développement)

// ------------------------------------------------------------
// v3.35 (0& oct 2010)
//  * Tableaux locaux
// v3.34 (03 fev 2010)
//  * Correctif MemoCompacte et SortWD (avec Memos)
// v3.32 (23 jan 2010)
//  + Ajout CRC sur données d'entrée, si :bCRCInput est activé
//  + Ajout cTxtVersTxt pour importer les données d'un autre objet cTxt ou pour le concatener
//  * Fix AddLine() si :nbCols < MesParametres..Occurrence
// v3.31 (22 jan 2010)
//  o Optimisation methode :Insert() avec TableauInsereLigne
//  + Ajout cTxtVersTxt pour importer les données d'un autre objet cTxt ou pour le concatener
//  + Ajout TxtVersHtml pour créer une table HTML
//  + Ajout AssocVersTxt pour copier un tableau associatif
//  + Ajout AddLine(col1,col2,....) compatible avec TableauAjouteLigne
//  + Ajout Parametre TxtVersTable() pour colonne de decalage
//  * IniVersTxt() pouvait encore poser probleme
// v3.30 (11 jan 2010)
//  - Renommage methode :Cell() en :CellGet() pour éviter les warnings redondants de Windev 15
//  - SortWD() probleme premier parametre pour tri multi colonne
//  * Premier() si vide, pour éviter les problemes dans certains cas
//  * Retrait warnings de parametre non utilisés sur fonctions de type zone mémoire
// v3.25 (05 jan 2010)
//  * IniVersTxt_Linux, retrait espaces des valeurs si ini du type "key = value"
// v3.24 (31 dec 2009)
//  * Retrait lignes vides de IniVersTxt
// v3.23 (20 juin 2008)
//  * Bug IniLit si derniere ligne
//  + Ajout IniVersTxt_Linux car IniLit ne fonctionne pas correctement sous linux
// v3.22 (19 mai 2008)
//  o IniVersTxt optimisé grace à l'API GetPrivateProfileSection (au moins 2x plus rapide)
//  o IniLit et IniEcrit optimisés grace à TableauCherche (attention les [SECTIONS] sont maintenant
forcées en majuscules)

//  o GetTxt() optimisé avec TableauVersChaîne (Windev 12 uniquement)
// v3.21 (10 mai 2008)
//  * Bug IniEcrit
```

```
// v3.20 (17 avr 2008)
//  + TxtVersExcel
// v3.11 (8 avr 2008)
//  * IniVersTxt :nModeMultitache relayé (pour wdscript linux)
// v3.10 (3 fév 2008)
//  * TxtVersTable, pas d'affectation si la table n'a pas assez de colonnes
//  * Bugfix GetEntete (v3.00)
//  + Ajout de la propriété bCSVQuotes pour prendre en compte les guillements pour chaque cellule
//  + DropDoubles gere maintenant plus de 2 doublons, à tester
//  + FiltreByLine, FiltreByCol renvoie le nouveau nombre de ligne
//  o Optimisation de TraceTxt sur une plage (colonnage correct), ajout des parametres facultatifs
ColMin, ColMax

//  o Typage des parametres de methodes, Ajout des commentaires de méthodes
// v3.00 (6 déc 2007)
//  * FindInCol() renvoie faux si la colonne ou la ligne mini n'existe pas
//  o Optimisation de Find avec TableauCherche lorsque cela est possible
//  o Fonction Line(L=:Encours,_ColDeb=1,_ColFin=:nbCols) maintenant sans parametre possible
//  * Sort() renvoie faux si la colonne demandée n'existe pas
//  + Support de la fonction Jauge() avec le membre :bJauge
//  o Optimisation du tri avec TableauTrie (50x plus rapide) (v10 mini)
//  o Optimisation des suppressions avec TableauSupprimeLigne (2x plus rapide) (v10 mini)
//  o AddCol(sFill=EOT) accepte un paramètre pour remplir la colonne
//  o Fonction Cell(L=:Encours,C=1) maintenant sans parametre possible
//  o Fonction Del(L=:Encours) maintenant sans parametre possible
//  o AddCol renvoie le nombre de colonnes
//  * MemoDel Renumerotation automatique
//  * Bug Fonction Precedent() (:endehors au lieu de :encours)
//  * Resize(), probleme redimensionnement colonnes / :nbcol
//  x StartDelete renommé en DeleteOptBegin(), EndDelete renommé en DeleteOptEnd() pour améliorer
mnémonique et autocomplétion dans le code "Endehors"

// v2.91 (29 Sep 2006)
//  * Premier(), Endehors à vrai si 1 seule ligne et vide
//  * Dernier() si vide renvoie "" et Endehors à vrai
//  + Fonction Cell(L,C=1) Pour accéder aux cellules au lieu de :Cell[L,C] qui peut provoquer une erreur
//  + Ajout param Fast pour DropDoubles (nécessite un tri)
//  + Ajout param nLastCol pour DropDoubles
// v2.90 (10 Jul 2006)
//  + Fonction DropDoubles pour supprimer les doublons
//  + Fonction GetCols comme GetTxt avec Colonnes min et max
//  + Fonction TxtVersFormatSpec pour exporter dans un fichier sans délimiteur en fournissant le format
de chaque colonne

// v2.85 (22 Feb 2006)
//  + ParseStream pour effacer au fur et à mesure la chaine source (reduit utilisation mémoire)
// v2.81 (22 Feb 2006)
//  + bSansEspace pour ignorer les espaces dans les colonnes
// v2.80 (20 Aug 2005)
//  * Fonction AddTxt, inversion SeparateursX/Y
//  + Récup. de l'entete dans AddTxt si vide
//  + Ajout Fonctions FindInLine et FindInHead
// v2.74 (18 Aug 2005)
//  + Ajout Fonction Positionne(_L) pour modifier la position en cours
// v2.73 (12 Aug 2005)
//  + Ajout Fonction ColOf pour recup sans exception d'une case (Col, Ligne) ou dans la ligne en cours
(Col)

// v2.72
//  + Ajout de l'option Merge pour IniVersTxt
// v2.71
//  + Ajout Option bIgnoreLignesVides (lors du parsing ou de l'ajout)
// v2.7
//  + Ajout Fonctions compatibles Ini (IniLit, IniEcrit)
//  x Pb bOnlyOneSeparatorX < Retiré (utilisez :nFixedColumns)
//  * Bug ExtractCol avec :nFixedColumns
//  * GetKeyValue (Option SansCasse par défaut)
//  * TraceTxt renvoie une valeur pour le débogage
// v2.6
//  + Ajout de donnees perso (MemoSet, MemoGet, MemoDel, MemoAdd, MemoMove, MemoCompacte)
//  + Ajout du SortQuick (Tony Haere Quick Sort (pas plus rapide pour le moment))
//  + Ajout de SwapLine (nécessaire pour le SortQuick)
//  + Ajout Commande TrimCol (Retrait Espaces)
//  + Ajout TxtVersArbre
```

```
//  + Ajout params à TraceTxt (Comment. / Plage)
//  + MemTrie avec Sens et Element
//  o Correction et Amélioration RegVersTxt
//  o Optimisation ExtractCol et gestion nFixedColumns (méthode la plus utilisée) (3-5%)
//  o Optimisation Sort (amélioration de 10% - opt. dépend de la taille des colonnes à déplacer)
//  o Optimisation ReverseLines (amélioration de 35-40% - idem)
// v2.51
//  * Bug Premier() si vide
// v2.5
//  + Ajout Méthode Move(Orig,Dest)
//  + Ajout Possibilité suppression Key dans SetKeyValue si Value vide (non par défaut)
//  * Bug Parse() avec données vide créait 1 ligne
//  * Bug Insert() ne fonctionnait pas
// v2.41
//  * Bug :ReverseLines() ne fonctionnait pas
//  o TraceTxt() > Alignement des EOT
// v2.4
//  * Bug apres :Resize() :nbLine non affecté
// v2.3
//  + Mode Nombre Colonnes Fixe ex: Pour Fichiers INI tj. à 3
//  + Ajout IniVersTxt, TxtVersIni
//  + Ajout RegVersTxt
// v2.2
//  + Ajout AddCol
//  + Ajout Resize
//  * Bug Start/EndDelete
//  + Amélioration SetKeyValue (Ajout si Key non trouvé)
// v2.1
//  + Ajout MemAjoute, MemModifie, MemSupprime, MemRécupère, MemValClé, MemOccurrence, MemTrie,
MemRecherche, MemEnCours (pour conversion rapide du code)

//  + Ajout FindInMultiCol
//  + Ajout Premier, Dernier, Suivant, Precedent et des propriétés EnCours et EnDehors
//  + Ajout TxtVersTable et TableVersTxt
// v2.0
//  + Ajout TraceTxt
//  + Ajout Fonctions Filtres
//  + Ajout Fonctions Fill
//  + Ajout GetMaxColLen
//  + Ajout SetKeyValue
//  + Ajout ExpandCol
//  + Ajout Clear
//  + Ajout TxtVersZMem
//  + NOM_JAUGE
//  - Renom zMemVersTxt
//
//  + Ajout d'un tableau d'entete correspondant au nb de colonnes
//  o Remplissage en EOT des colonnes ajoutées (pris en compte dans GetTxt)
//  + Ajout du mode multitache
//  + Ajout GetKeyValue
//  + Ajout SetLine

rMemo est une structure
    nLin est entier
    nCol est entier

    Data est une chaîne
FIN

cTxt est une classe

    PRIVÉ

        bDeleting     est booléen
        nFirstLineDel est entier

        nMemoCount    est entier

    PUBLIC CONSTANT

        nbLines est entier
        nbCols  est entier

        EnCours est entier
```

```
    EnDehors est booléen

    nMemEnCours  est entier
    bMemEnDehors est booléen
    bMemTrouve   est booléen

    //pour ParseStream
    bCropSource  est booléen

    bLinux est boolean

PUBLIC

    // Entete (Nom des colonnes)
    Head est un tableau local de 1 chaîne
    //Head est un tableau dynamique (Ancien mode)

    // Donnees
    Cell est un tableau local de 1 par 1 chaîne
    //Cell est un tableau dynamique (Ancien mode)

    // Donnees Suppl (Optionel)
    Memo est un tableau dynamique

    sSeparatorX est chaîne // Défaut = TAB
    sSeparatorY est chaîne // Défaut = RC

    //---------------------------------------
    // Options :
    nFixedColumns      est entier

    //Mode de multitache pendant les traitements longs
    nModeMultitache    est entier

    //Numero de ligne pour Entete
    nEnteteLine        est entier

    //Nb de lignes à ignorer
    nbEnteteLines       est entier

    //Ne pas ajouter les lignes vides
    bIgnoreLignesVides est booléen

    //Ex: ";" dans un fichier INI
    sIgnoredLinesComment est chaîne

    //Retrait des lignes vides à la fin du fichier
    bDelLastEmptyLines est booléen

    //Utilisation donnees optionelles
    bUseMemo est booléen

    //Ignorer les espaces en fin de données des colonnes
    bSansEspace est booléen

    //Prendre en compte/Utiliser les guillemets pour les cellules, permet le multiligne et les
    tabulations (à valider)

    bCSVQuotes est booléen
    //---------------------------------------

    //Optionnel : Nom du champ Jauge pour opérations longues
    NOM_JAUGE est chaîne

    //Utiliser la fonction Jauge() (Champ en barre d'état par défaut)
    bJaugeWD est booléen

    //Précision de la jauge (peut accélérer legerement le traitements si >)
    nJaugeMinStep est entier

    // Chaine de caractères définissant le nom de la zone mémoire en chaine (pour compatibilité)
    sMem    est une chaîne

    //CRC du texte parsé à l'origine (Parse()) pour comparaison rapide d'objets en cache
```

```
        nCRCInput est entier sans signe
        bCRCInput est boolean
        sID_Input est chaîne
        // pour certaines entrées (INI par ex), cet identifiant peut eviter le calcul de CRC

    FIN
```

Constructeur

```
PROCEDURE Constructeur(_Data="",_SeparatorY=RC,_SeparatorX=TAB,_bUseEntete=Faux,_sMem="")

:sSeparatorY = _SeparatorY
:sSeparatorX = _SeparatorX

:Head = allouer un tableau de 1 chaîne
:Cell = allouer un tableau de 1 par 1 chaîne
:Memo = allouer un tableau de 0 rMemo

:nJaugeMinStep = 100

:nModeMultitache = 0
:bDelLastEmptyLines = Vrai

:nFixedColumns = 0 // Désactivé

// Chaine de caractères définissant le nom de la zone mémoire en chaine (pour compatibilité)
:sMem = _sMem

SI _Data <> "" ALORS
    :Parse(_Data)
FIN

// Retrait warning
SI _bUseEntete ALORS
FIN
```

```
:bLinux=Vrai
```

Destructeur

```
PROCEDURE Destructeur()

//si :cell..Occurrence > 0 ALORS
//  libérer :Cell
//FIN
//SI :Head..Occurrence > 0 ALORS
//  libérer :Head
//FIN
```

Méthode ExtractLines

```
// Résumé : Convertit la chaine :Data en tableau de lignes grace au séparateur Vertical (RC par défaut)
les colonnes sont ensuite extraites par ExtractCols

// Syntaxe :
// [ <Résultat> = ] ExtractLines (<_Data> est chaîne)
//
// Paramètres :
//  _Data (chaîne) : <indiquez ici le rôle de _Data>
// Valeur de retour :
//  entier : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
```

```
//
PROCEDURE PRIVÉE VIRTUELLE ExtractLines(_Data est chaîne)
//cTxt

L est entier
S est chaîne
E est entier

nD,nF,nlS,nlC sont entiers

SI :bCSVQuotes ALORS
    :ExtractCSV(_Data)
FIN

SI _Data<>"" ALORS
    :nbLines = ChaîneOccurrence(_Data,:sSeparatorY)+1
SINON
    :nbLines = 0
FIN


SI :nEnteteLine OU :nbEnteteLines ALORS
    E = :nEnteteLine
    SI E>0 ALORS
        Dimension(:Head,1)
        :Head[1] = ExtraitChaîne(_Data,E,:sSeparatorY)
        :nbEnteteLines += 1
    FIN
    :nbLines -= :nbEnteteLines
FIN

:JaugeMinMax(0,:nbLines*2)

Dimension(:Cell,:nbLines,1)

nlS = Taille(:sSeparatorY)
nlC = Taille(:sIgnoredLinesComment)

nD=1

POUR L=1 _A_ :nbEnteteLines
    nF=Position(_Data,:sSeparatorY,nD)
    nD=nF+nlS
FIN

E=:nbEnteteLines

POUR L=1 A (:nbLines-E)

    nF=Position(_Data,:sSeparatorY,nD)

    SI nF = 0 ALORS
        S = _Data[[nD A]]
        :nbLines = L-E
    SINON
        S = _Data[[nD A nF-1]]
    FIN

    SI S="" ET :bIgnoreLignesVides ALORS
        E++
    SINON
        :Cell[Max(1,L-E),1] = S

        SI nlC ALORS
            SI S[[A nlC]] = :sIgnoredLinesComment ALORS
                E++
            FIN
        FIN
    FIN

    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)

    nD=nF+nlS
```

```
    //Utilisé avec ParseStream, on reduit l'utilisation mémoire
    SI :bCropSource ET nD>1000000 ALORS
        _Data=_Data[[nD+1 A]]
        nD=1
    FIN

FIN

SI :bDelLastEmptyLines ET :nbLines>0 ET PAS :bIgnoreLignesVides ALORS
    TANTQUE :Cell[:nbLines,1] = ""
        SI :nbLines=1 ALORS SORTIR
        :nbLines--
    FIN
FIN

:nbCols = 1
Dimension(:Cell,:nbLines,:nbCols)

RENVOYER :nbLines
```

Méthode ExtractCols

```
// Résumé : Convertit des lignes en multicolonne grace au séparateur horizontal (TAB par défaut)
// Syntaxe :
//[ <Résultat> = ] ExtractCols ( [<_LineMin> est entier [, <_LineMax> est entier]])
//
// Paramètres :
//  _LineMin (entier - valeur par défaut=1) : Première ligne
//  _LineMax (entier) :nbLines) : Dernière ligne
// Valeur de retour :
//  entier :    Nombre de colonnes
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PROTÉGÉE ExtractCols(_LineMin est entier=1,LOCAL _LineMax est entier=:nbLines)

L,C est entier

POUR L=_LineMin A _LineMax

    :ExtractCol(L,Faux)

    SI :bCSVQuotes ALORS
        POUR C=:nbCols A 1 PAS -1
            SI Position(:Cell[L,C],"¤") ALORS
                :Cell[L,C]=Remplace(:Cell[L,C],"¤TAB¤",:sSeparatorX)
                :Cell[L,C]=Remplace(:Cell[L,C],"¤RC¤",:sSeparatorY)
            FIN
        FIN
    FIN

    SI PAS (L modulo :nJaugeMinStep) :JaugeSet(:nbLines+L)

FIN

:JaugeSet(0)

RENVOYER :nbCols
```

Méthode Parse

```
// Résumé : Transforme :Data en Tableau (méthode principale)
// Syntaxe :
//[ <Résultat> = ] Parse (<_Data> est chaîne)
//
// Paramètres :
//  _Data (chaîne) : Chaine à parser
// Valeur de retour :
//  entier :    //  Aucune
//// Exemple :
// oTxt:sSeparatorX = ";"
// oTxt:Parse(fChargeTexte(sCheminFichier))
//
```

```
PROCEDURE PUBLIQUE Parse(_Data est chaîne)

SI :nbLines ALORS

    //Optimisation via CRC
    SI :bCRCInput ALORS
        nCRC est entier sans signe
        nCRC = sCalculeCrc32(_Data)
        SI nCRC _ET_ :nCRCInput = nCRC ALORS
            RENVOYER :nbLines
        FIN
        :nCRCInput = nCRC
    FIN

    :Clear()
FIN

SI :bCRCInput ALORS
    :nCRCInput = nCRC
    SI nCRC=0 ALORS :nCRCInput = sCalculeCrc32(_DATA)
FIN

:ExtractLines(_Data)
:ExtractCols()
:ExtractEntete()

RENVOYER :nbLines
```

Méthode Reset

```
// Résumé : Libère le tableau dynamique et réalloue un tableau minimal comme à l'origine.
:nblines,:nbCols,:nMemoCount,:nFixedColumns,:nbEnteteLines remis 0

// Syntaxe :
//Reset ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Reset()

:nFixedColumns = 0
:nbEnteteLines = 0

:nbLines = 0
:nbCols  = 0
TableauSupprimeTout(:Head)
TableauSupprimeTout(:Cell)
Dimension(:Head,1)
Dimension(:Cell,1,1)

//SI TableauInfo(:Cell,tiNombreTotal) > 0 ALORS
//  libérer :Cell
//FIN

libérer :Memo
:nMemoCount = 0

//:Head = allouer un tableau de 1 chaîne
//:Cell = allouer un tableau de 1 par 1 chaîne
:Memo = allouer un tableau de 0 rMemo
```

Méthode Find

```
// Résumé : Recherche une valeur dans le tableau
// Syntaxe :
//[ <Résultat> = ] Find (<_sRech> est chaîne, <Line>, <Col> [, <_nLineMin> est entier [, <_nLineMax> est entier [, <_nColMin> est entier [, <_nColMax> est entier [, <_Options> est entier]]]]])

//
// Paramètres :
//  _sRech (chaîne) : <indiquez ici le rôle de _sRech>
//  Line : Variable à passer par adresse pour récupérer la ligne de la cellule trouvée
//  Col : Variable à passer par adresse pour récupérer la colonne de la cellule trouvée
//  _nLineMin (entier - valeur par défaut=1) : Ligne de début de la recherche
//  _nLineMax (entier) :nbLines) : Ligne de fin de la recherche
//  _nColMin (entier - valeur par défaut=1) : Colonne de début de la recherche
//  _nColMax (entier) :nbCols) : Colonne de fin de la recherche
//
_Options (entier - valeur par défaut=0) : Mode de recherche, à l'identique (donc chaine uniquement) par défaut (SansCasse/MotComplet, combinable)

// Valeur de retour :
//  booléen :     Résultat trouvé ? Si oui, les paramètres par adresse Line et Col sont affectés
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Find(_sRech est chaîne,Line,Col,_nLineMin est entier=1,LOCAL _nLineMax est entier=:nbLines,_nColMin est entier=1,LOCAL _nColMax est entier=:nbCols,_Options est entier=0)

SI _nLineMax=0 ALORS
    _nLineMax=:nbLines
FIN

SI _nColMax=0 ALORS
    _nColMax=:nbCols
FIN

bRechEffectuee est booléen
L,C sont entiers
SI _Options=0 ET _nLineMax=:nbLines ALORS
    SI _nColMin=_nColMax ALORS
        L=TableauCherche(:Cell,tcLinéaire,_nColMin,_sRech,_nLineMin)
        bRechEffectuee=Vrai
    SINON SI _nColMin=1 ET _nColMax=:nbCols ALORS
        L=TableauCherche(:Cell,tcLinéaire,_sRech,_nLineMin)
        bRechEffectuee=Vrai
    FIN
    SI L>=0 ALORS
        Line=L
        RENVOYER Vrai
    SINON
        SI bRechEffectuee ALORS
            RENVOYER Faux
        FIN
    FIN
FIN

POUR L=_nLineMin A _nLineMax
    POUR C=_nColMin A _nColMax
        SELON _Options
            CAS 0
                //Recherche Stricte
                SI :Cell[L,C] = _sRech ALORS
                    Line = L
                    Col  = C
                    RENVOYER Vrai
                FIN
            CAS SansCasse
                //Recherche sans attention aux Min/Maj et Espaces
                SI :Cell[L,C] ~= _sRech ALORS
                    Line = L
                    Col  = C
                    RENVOYER Vrai
                FIN
            AUTRES CAS
                //MotComplet/DepuisDebut/DepuisFin (ou combinaisons)
```

```
                SI Position(:Cell[L,C],_sRech,1,_Options) > 0 ALORS
                    Line = L
                    Col  = C
                    RENVOYER Vrai
                FIN
            FIN
        FIN
FIN

RENVOYER Faux
```

Méthode SortOld

```
PROCEDURE SortOld(_ByCol est entier=1,_bSensPositif est booléen=Vrai,_bTriNumerique est booléen=Faux,
_sFormatNum est chaîne="012d")

SI _ByCol>:nbCols ALORS
    RENVOYER Faux
FIN

:JaugeMinMax(0,:nbLines*2)

// Création de la zone mémoire temporaire unique pour le tri
//-------------------------------------------------------------
z est entier; zMem est chaîne
BOUCLE
    zMem = "zcTab"+NumériqueVersChaîne(z,"06d")
    SI PAS WL.MemExiste(zMem) ALORS SORTIR
    z++
FIN
WL.MemCrée(zMem)
//-------------------------------------------------------------

L,C,D sont entiers

POUR L=1 _A_ :nbLines
    SI _bTriNumerique ALORS
        WL.MemAjoute(zMem,"N"+NumériqueVersChaîne(Val(:Cell[L,_ByCol]),_sFormatNum),L)
    SINON
        WL.MemAjoute(zMem,:Cell[L,_ByCol],L)
    FIN
    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
FIN
WL.MemTrie(zMem,_bSensPositif)

TableauTmp est un tableau dynamique = allouer un tableau de :nbLines par :nbCols chaînes

TableauCopie(:Cell,TableauTmp)

////si versionwindev < v10
//POUR L=1 a :nbLines
//  POUR C=1 a :nbCols
//      TableauTmp[L,C] = :Cell[L,C]
//  FIN
//FIN

bResult est un booléen = Vrai

POUR L=1 _A_ :nbLines

    D = Val(MemRecupère(zMem,L))
    SI D>0 ALORS
        POUR C=1 _A_ :nbCols
            :Cell[L,C] = TableauTmp[D,C]
        FIN
        SI :bUseMemo ALORS
            :MemoMove(D,0,L)
        FIN
    SINON
        //Rollback
        POUR L=1 A L
            POUR C=1 _A_ :nbCols
                :Cell[L,C] = TableauTmp[L,C]
            FIN
```

```
        FIN
        bResult = Faux
    FIN

    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(:nbLines+L)

FIN

WL.MemSupprimeTout(zMem)

libérer TableauTmp

:JaugeSet(0)

RENVOYER bResult
```

Méthode Add

```
// Résumé : Ajoute une ligne multicolonne, colonnes extraites automatiquement
// Syntaxe :
//[ <Résultat> = ] Add (<_Ligne> est chaîne)
//
// Paramètres :
//  _Ligne (chaîne) : <indiquez ici le rôle de _Ligne>
// Valeur de retour :
//  booléen :      <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Add(_Ligne est chaîne)

:nbCols = Max(:nbCols,:nFixedColumns,1)
:nbLines++

Dimension(:Cell,:nbLines,:nbCols)
:SetLine(:nbLines,_Ligne)

RENVOYER Vrai
```

Méthode Del

```
// Résumé : Supprime une ligne
// Syntaxe :
//[ <Résultat> = ] Del ( [<_nLine> est entier])
//
// Paramètres :
//  _nLine (entier) : Indice de la ligne
// Valeur de retour :
//  booléen :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Del(_nLine est entier=:EnCours)

SI _nLine < 1 OU _nLine > :nbLines ALORS
    RENVOYER Faux
FIN

SI :bDeleting ALORS
    :Cell[_nLine,1] = "[%¤'DELETED'¤%]"
    SI :nFirstLineDel = 0 ALORS
        :nFirstLineDel = _nLine
    SINON
        :nFirstLineDel = Min(_nLine,:nFirstLineDel)
    FIN
SINON

    // Windev < v10
    //  :Cell[_nLine,1] = "[%¤'DELETED'¤%]"
    //  :nFirstLineDel = _nLine
    //  :DeleteOptEnd()

    TableauSupprimeLigne(:Cell,_nLine)
```

```
    :nbLines--

    // Suppression Memos de la ligne
    SI :bUseMemo ALORS
        :MemoDel(_nLine)
    FIN

FIN

RENVOYER Vrai
```

Méthode Insert

```
// Résumé : Insere une ligne avant l'indice spécifié, colonnes extraites automatiquement
// Syntaxe :
//[ <Résultat> = ] Insert (<_AvantLigne> est entier, <_Chaine> est chaîne)
//
// Paramètres :
//  _AvantLigne (entier) : <indiquez ici le rôle de _AvantLigne>
//  _Chaine (chaîne) : <indiquez ici le rôle de _Chaine>
// Valeur de retour :
//  booléen :     <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Insert(_AvantLigne est entier,_Chaine est chaîne)

SI _AvantLigne > :nbLines ALORS
    RENVOYER :Add(_Chaine)
FIN

SI _AvantLigne < 0 ALORS
    RENVOYER Faux
FIN

:JaugeMinMax(0-:nbLines,0-_AvantLigne)

L,C est entier

:nbLines++
SI :bUseMemo ALORS
    Dimension(:Cell,:nbLines,:nbCols)

    POUR L=:nbLines _A_ (_AvantLigne+1) PAS -1
        POUR C=1 _A_ :nbCols
            :Cell[L,C] = :Cell[L-1,C]
        FIN

        SI :bUseMemo ALORS
            :MemoMove(L-1,0,L)
        FIN

        SI (L modulo :nJaugeMinStep = 0) :JaugeSet(0-L)
    FIN
SINON
    TableauInsèreLigne(:Cell,1,"")
FIN

:Cell[_AvantLigne,1] = _Chaine
:ExtractCol(_AvantLigne)

:JaugeSet(0-:nbLines)

RENVOYER Vrai
```

Méthode Line

```
// Résumé : Récupère la ligne complète à l'indice spécifiée, les numéros de colonnes de début et fin sont
facultatives

// Syntaxe :
//[ <Résultat> = ] Line ( [<_L> est entier [, <_ColDeb> est entier [, <_ColFin> est entier]]])
//
// Paramètres :
//  _L (entier) : Indice de la ligne
//  _ColDeb (entier - valeur par défaut=1) : Colonne de début
//  _ColFin (entier) :nbCols) : Colonne de fin
// Valeur de retour :
//  chaîne :    <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Line(_L est entier=:EnCours,_ColDeb est entier=1,LOCAL _ColFin est entier=:nbCols)

// Renvoie la ligne _L

SI _L < 1 OU _L > :nbLines ALORS
    RENVOYER EOT
FIN

sRes est chaîne
C    est entier

POUR C=_ColDeb _A_ _ColFin
    SI :Cell[_L,C] = EOT ALORS
        SORTIR
    SINON
        sRes += :Cell[_L,C]+:sSeparatorX
    FIN
FIN

// Retrait du dernier séparateur horizontal
SI Sres <> "" ALORS
    Sres = Sres[[A Taille(Sres)-Taille(:sSeparatorX)]]
FIN

RENVOYER Sres
```

Méthode ExtractCol

```
// Résumé : Découpe la cellule :Cell[_Ligne,1] dans sa ligne
// Syntaxe :
//ExtractCol (<_Ligne> est entier [, <_bFillNewCol> est booléen])
//
// Paramètres :
//  _Ligne (entier) : Indice de ligne
//  _bFillNewCol (booléen - valeur par défaut=1) : Remplit de EOT les nouvelles colonnes
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
//  _bSetEntete (booléen - valeur par défaut=0) : <indiquez ici le rôle de _bSetEntete>
PROCEDURE PRIVÉE VIRTUELLE ExtractCol(_Ligne est entier,_bFillNewCol est booléen=Vrai)


C, nlS, nTotCol sont entiers
sL sont chaînes

nlS = Taille(:sSeparatorX)

sL = :Cell[_Ligne,1]

SI :nFixedColumns > 0 ALORS

    nPosData est entier

    //MODE :nFixedColumns COLONNES MAX (la derniere colonne peut contenir des SeparatorX)
    nTotCol = :nFixedColumns
```

```
    SI :nbCols <> nTotCol ALORS
        :nbCols = nTotCol
        Dimension(:Cell,:nbLines,:nbCols)
        Dimension(:Head,:nbCols)
    FIN

    POUR C=1 A nTotCol
        :Cell[_Ligne,C] = ExtraitChaîne(sL,C,:sSeparatorX)
        //nLenKey+=Taille(:Cell[_Ligne,C])+nlS
        SI C=:nFixedColumns ALORS
            //La derniere colonne recoit tout le reste ex: sec,key,xxx,yyy >> xxx,yyy
            nPosData=PositionOccurrence(sL,:sSeparatorX,:nFixedColumns-1)+nLs
            SI :bSansEspace ALORS
                :Cell[_Ligne,:nFixedColumns] = SansEspace(sL[[nPosData A]])
            SINON
                :Cell[_Ligne,:nFixedColumns] = sL[[nPosData A]]
            FIN

//          SI :bSansEspace ALORS
//
:Cell[_Ligne,:nFixedColumns] = SansEspace(sL[[Taille(:Line(_Ligne,1,:nFixedColumns-1))+nlS+1 A ]])
//          SINON
//              :Cell[_Ligne,:nFixedColumns] = sL[[Taille(:Line(_Ligne,1,:nFixedColumns-1))+nlS+1 A ]]
//          FIN
            SORTIR
        FIN
    FIN

SINON

    //MODE NORMAL MULTICOLONNE
    nTotCol = ChaîneOccurrence(sL,:sSeparatorX) + 1
    SI nTotCol > :nbCols ALORS
        Dimension(:Cell,:nbLines,nTotCol)
        Dimension(:Head,nTotCol)
        SI _bFillNewCol ALORS
            // On remplit de EOT l(a/es) nouvelle(s) colonne(s) ajoutée(s)
            :FillCols(:nbCols+1,nTotCol,EOT)
        SINON
            // On remplit de EOT uniquement les lignes précédentes
            :FillCols(:nbCols+1,nTotCol,EOT,1,_Ligne-1)
        FIN
        :nbCols = nTotCol
    SINON
        nTotCol=:nbCols
    FIN

    POUR C=nTotCol+1 A nTotCol
        :Cell[_Ligne,C] = EOT
    FIN
    POUR C=nTotCol A 1 PAS -1
        SI :bSansEspace ALORS
            :Cell[_Ligne,C] = SansEspace(ExtraitChaîne(sL,C,:sSeparatorX))
        SINON
            :Cell[_Ligne,C] = ExtraitChaîne(sL,C,:sSeparatorX)
        FIN
    FIN

    //  Pas plus rapide (A tester sans le ChaineOccurrence)
    //  nF,nD sont entiers = 1
    //  POUR C=1 A nTotCol-1
    //      nF=Position(sL,:sSeparatorX,nD)
    //      :Cell[_Ligne,C] = sL[[nD a nF-1]]
    //      nD=nF+nlS
    //  FIN
    //  :Cell[_Ligne,nTotCol] = sL[[nD a]]

FIN
```

Méthode DeleteOptBegin

```
// Résumé : Optimise la suppression en cas d'utilisation multiple (raye la ligne au lieu de supprimer
réellement, plus forcement utile apres Windev 10)

// Syntaxe :
// DeleteOptBegin ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//
// Exemple :
// cTab:DeleteOptBegin()
// POUR L=1 a cTab:nbLines
//   SI cTab:Cellule[L,1] = "TRUC" ALORS
//      cTab:Del(L)
//   FIN
// FIN
// cTab:DeleteOptEnd()
//
PROCEDURE PUBLIQUE DeleteOptBegin()

:bDeleting     = Vrai
:nFirstLineDel = 0
```

Méthode DeleteOptEnd

```
// Résumé : supprime les lignes rayées
// Syntaxe :
// DeleteOptEnd ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//
// Exemple :
// DeleteOptBegin()
// ...
// Del()
// Del()
// ...
// DeleteOptEnd()

PROCEDURE PUBLIQUE DeleteOptEnd()

//:JaugeMinMax(:nFirstLineDel,:nbLines)

L est entier
C est entier
bDel est booléen
nbDeleted est entier
nTotLig,nTotCol est entier

L=:nFirstLineDel
BOUCLE
    SI L > :nbLines OU L<=0 ALORS
        SORTIR
    FIN

    SI :Cell[L,1] = "[%¤'DELETED'¤%]" ALORS
        nbDeleted++
        bDel=Vrai

        // Suppression Memos de la ligne
        SI :bUseMemo ALORS
            :MemoDel(L)
        FIN

    SINON
        bDel=Faux
    FIN
```

```
    SI (PAS bDel) ET nbDeleted>0 ALORS
        nTotCol=:nbCols
        POUR C=1 A nTotCol
            :Cell[L-nbDeleted,C] = :Cell[L,C]
        FIN
    FIN

    L++

    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
FIN

nTotLig=:nbLines
POUR L=(:nbLines-nbDeleted+1) A nTotLig
    :SetLine(L,"")
FIN
:nbLines-=nbDeleted

SI :nbCols = 0 ALORS
    SI EnModeTest() Trace("cTxt:EndDelete()","Attention :nbCols = 0 !")
FIN
Dimension(:Cell,:nbLines,:nbCols)

:bDeleting     = Faux
:nFirstLineDel = 0

//:JaugeMinMax(0,:nbLines)
```

Méthode ReverseLines

```
// Résumé : Inverse l'ordre des lignes du tableau
// Syntaxe :
// ReverseLines ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE ReverseLines()

//A FAIRE:Verifier avec entete
L,C sont entiers

tLineTmp est un tableau dynamique = allouer un tableau de :nbCols chaînes
nTotLig est entier=PartieEntière(:nbLines/2)
nTotCol est entier=:nbCols

POUR L=1 A nTotLig

    // Appeler :SwapLine() réduit de 20% les performances (bizarre)

    POUR C=1 A nTotCol
        tLineTmp[C] = :Cell[L,C]
        :Cell[L,C] = :Cell[:nbLines-L+1,C]
        :Cell[:nbLines-L+1,C] = tLineTmp[C]
    FIN

    SI :bUseMemo ALORS
        :MemoMove(L,0,:nbLines+1)
        :MemoMove(:nbLines-L+1,0,L)
        :MemoMove(:nbLines+1,0,L)
    FIN

FIN

libérer tLineTmp
```

Méthode GetTxt

```
// Résumé : Renvoie le tableau complet sous forme de chaine, séparés par les séparateurs définis
// Syntaxe :
//[ <Résultat> = ] GetTxt ( [<bWithEntete> est booléen])
//
// Paramètres :
//  bWithEntete (booléen - valeur par défaut=0) : < indiquez ici le rôle de bWithEntete >
// Valeur de retour :
//  chaîne :       <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE GetTxt(bWithEntete est booléen=Faux)

sRes est chaîne

SI bWithEntete ALORS
    Sres += :GetEntete()+:sSeparatorY
FIN

SI :bCSVQuotes ALORS
    sRes+=TableauVersCSV(:Cell,:sSeparatorY)
SINON
    sRes+=TableauVersChaîne(:Cell,:sSeparatorY,:sSeparatorX)
FIN

//Windev 10:
//nTotLig=:nbLines
//POUR L=1 a nTotLig
//  Sres += :Line(L)+:sSeparatorY
//FIN
//
//SI Sres <> "" ALORS
//  Sres = Sres[[a Taille(Sres)-Taille(:sSeparatorY)]]
//FIN

RENVOYER sRes
```

Méthode FormatLine

```
// Résumé : Renvoie une chaine formatée avec le séparateur horizontal (multi paramètres)
// Syntaxe :
//[ <Résultat> = ] FormatLine ([...])
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  chaîne :    <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// oTxt:Add(oTxt:FormatLine(A,B,C))
//
//FONCTION PUBLIQUE
FormatLine(_C1,_C2=EOT,_C3=EOT,_C4=EOT,_C5=EOT,_C6=EOT,_C7=EOT,_C8=EOT,_C9=EOT,_C10=EOT,_C11=EOT,_C12=EOT
,_C13=EOT,_C14=EOT,_C15=EOT,_C16=EOT,_C17=EOT,_C18=EOT,_C19=EOT,_C20=EOT)

FONCTION PUBLIQUE FormatLine(*)

sRes est chaîne
//sParam est une chaîne
//
//N est entier
//POUR N = 1 A 20
//  sParam={"_C"+N,indVariable}
//  SI sParam = EOT ALORS SORTIR
//  sRes += (sParam + :sSeparatorX)
//FIN

POUR N = 1 _A_ MesParamètres..Occurrence
    sRes += (MesParamètres[N] + :sSeparatorX)
FIN

sRes = sRes[[A Taille(sRes)-Taille(:sSeparatorX)]]
```

```
RENVOYER sRes
```

Méthode FindInCol

```
// Résumé : Recherche simplifiée, pas besoin de passer des paramètres en tant que variable de retour pour obtenir la ligne du résultat (pas la colonne)

// Syntaxe :
//[ <Résultat> = ] FindInCol (<_sRech> est chaîne, <_nCol> est entier [, <_Options> est entier [, <_nLineMin> est entier [, <_nLineMax> est entier]]])

//
// Paramètres :
//  _sRech (chaîne) : Chaine recherchée
//  _nCol (entier) : Numéro de la colonne pour la recherche
//  _Options (entier - valeur par défaut=0) : SansCasse/MotComplet
//  _nLineMin (entier - valeur par défaut=1) : Indice de début de la recherche
//  _nLineMax (entier) : Indice de fin de la recherche
// Valeur de retour :
//  entier :   Le Résultat est le numéro de ligne ou 0 si non trouvé
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE FindInCol(_sRech est chaîne,_nCol est entier,_Options est entier=0,_nLineMin est entier=1,_nLineMax est entier=:nbLines)

L,C sont entiers

SI C>:nbCols ALORS
    RENVOYER 0
FIN

SI _nLineMin>:nbLines ALORS
    RENVOYER 0
FIN

SI PAS :Find(_sRech,L,C,_nLineMin,_nLineMax,_nCol,_nCol,_Options) ALORS
    RENVOYER 0
FIN

RENVOYER L
```

Méthode JaugeMinMax

```
PROCEDURE PROTÉGÉE JaugeMinMax(_Min,_Max,_Val=0)

SI :bLinux ALORS RETOUR

QUAND EXCEPTION
    ExceptionActive()
    :NOM_JAUGE = ""
    RETOUR
FIN

SI :NOM_JAUGE <> "" ALORS
    {:NOM_JAUGE,indChamp}..BorneMin = _Min
    {:NOM_JAUGE,indChamp}..BorneMax = _Max
    {:NOM_JAUGE,indChamp}..Valeur = _Val
    :MultitachePerso()
SINON SI :bJaugeWD ALORS
    Jauge(_Val,_Max)
FIN
```

Méthode JaugeSet

```
PROCEDURE PROTÉGÉE JaugeSet(_Val)

SI :bLinux ALORS RETOUR

SI :NOM_JAUGE <> "" ALORS
    {:NOM_JAUGE,indChamp}..Valeur = _Val
SINON SI :bJaugeWD ALORS
    Jauge(_Val)
FIN

//test nécessaire en linux
SI :nModeMultitache>=0 ALORS
    :MultitachePerso()
FIN
```

Méthode MultitachePerso

```
PROCEDURE PROTÉGÉE MultitachePerso()

SI :bLinux ALORS RETOUR

SELON :nModeMultitache
    CAS -1
        //linux par ex.
    CAS 0
        Multitâche()
    CAS 1
        MultitâcheRepeint()
    CAS 2
        Multitâche(-1)
FIN
```

Méthode GetEntete

```
// Résumé : Renvoie la ligne d'entete
// Syntaxe :
//[ <Résultat> = ] GetEntete ( [<_nColDeb> est entier [, <_nColFin> est entier]])
//
// Paramètres :
//  _nColDeb (entier - valeur par défaut=1) : <indiquez ici le rôle de _nColDeb>
//  _nColFin (entier) : <indiquez ici le rôle de _nColFin>
// Valeur de retour :
//  chaîne :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE GetEntete(_nColDeb est entier=1,LOCAL _nColFin est entier=:nbCols)

sRes est chaîne
C    est entier

SI _nColFin=0 ALORS
    _nColFin=:nbCols
FIN

//le tableau :Head est mono dimension donc tiNombreLignes
_nColFin = Min(_nColFin,TableauInfo(:Head,tiDimension))

POUR C=_nColDeb A _nColFin
    SI :Head[C] = EOT ALORS
        SORTIR
    SINON
        sRes += :Head[C]+:sSeparatorX
    FIN
FIN

// Retrait du dernier séparateur horizontal
SI Sres <> "" ALORS
    Sres = Sres[[ A Taille(Sres)-Taille(:sSeparatorX)]]
FIN
```

```
RENVOYER Sres
```

Méthode ExtractEntete

```
// Résumé : Similaire à ExtractCol, mais pour l'entete (titres des colonnes par exemple)
// Syntaxe :
// ExtractEntete ()
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PROTÉGÉE ExtractEntete()

// Découpe :Cell[_Ligne,1] vers la ligne

C, nTotCol sont entiers
sL sont chaînes

SI TableauInfo(:Head,tiNombreTotal)>0 ALORS
    sL = :Head[1]
FIN

SI :bCSVQuotes ALORS
    sL=Remplace(sL,"¤TAB¤",:sSeparatorX)
FIN

//MODE NORMAL MULTICOLONNE

nTotCol = ChaîneOccurrence(sL,:sSeparatorX) + 1
SI nTotCol > :nbCols ALORS
    :Resize(:nbLines,nTotCol)
FIN

POUR C=:nbCols A 1 PAS -1
    :Head[C] = ExtraitChaîne(sL,C,:sSeparatorX)
FIN
```

Méthode SetLine

```
// Résumé : Affecte une ligne existante et en extrait les colonnes
// Syntaxe :
//[ <Résultat> = ] SetLine (<_Indice> est entier, <_Text> est chaîne)
//
// Paramètres :
//  _Indice (entier) : <indiquez ici le rôle de _Indice>
//  _Text (chaîne) : Texte à affecter à la ligne
// Valeur de retour :
//  booléen :   //  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE SetLine(_Indice est entier,_Text est chaîne)

// Suite à un problème de sélection très rapide,
// il faut attendre la fin du traitement précédent sinon la variable _Indice = 0
SI _Indice <= 0 OU _Indice > :nbLines ALORS RENVOYER Faux

:Cell[_Indice,1] = Remplace(_Text,:sSeparatorY,"¤RC¤")
:ExtractCol(_Indice)

RENVOYER Vrai
```

Méthode GetKeyValue

```
// Résumé : Methode similaire aux tableaux associatifs, inexistants à l'époque
// Syntaxe :
//[ <Résultat> = ] GetKeyValue (<_sKey> est chaîne [, <_nColRetour> est entier [, <_nOptions> est
entier]])

//
// Paramètres :
//  _sKey (chaîne) : Valeur de cellule recherchée dans la première colonne
//  _nColRetour (entier - valeur par défaut=2) : Colonne du Résultat à récupérer
//  _nOptions (entier - valeur par défaut=Sanscasse) : SansCasse ou non
// Valeur de retour :
//  Type chaine :  Contenu de la cellule de la colonne résultat
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE GetKeyValue(_sKey est chaîne,_nColRetour est entier=2,_nOptions est entier=SansCasse)

L est entier = :FindInCol(_sKey,1,_nOptions)
SI L > 0 ALORS
    RENVOYER :Cell[L,_nColRetour]
FIN

RENVOYER EOT
```

Méthode AddTxt

```
// Résumé : Ajoute du contenu au tableau existant, parsé automatiquement
// Syntaxe :
//AddTxt (<_Text> est chaîne)
//
// Paramètres :
//  _Text (chaîne) : <indiquez ici le rôle de _Text>
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE AddTxt(_Text est chaîne)

// Ajoute un nouveau bloc (une ou plusieurs lignes)
oTmpTxt est un cTxt("",:sSeparatorY,:sSeparatorX)
oTmpTxt:nFixedColumns        = :nFixedColumns
oTmpTxt:nModeMultitache      = :nModeMultitache
oTmpTxt:sIgnoredLinesComment = :sIgnoredLinesComment
oTmpTxt:bIgnoreLignesVides   = :bIgnoreLignesVides
oTmpTxt:bIgnoreLignesVides   = :bIgnoreLignesVides
SI :nbLines=0 ET :Head[1]="" ALORS
    oTmpTxt:nEnteteLine = :nEnteteLine
FIN
oTmpTxt:Parse(_Text)

SI oTmpTxt:Head[1]<>"" ALORS
    :Head[1] = oTmpTxt:GetEntete()
    :ExtractEntete()
    :nbEnteteLines = oTmpTxt:nbEnteteLines
FIN

L,C sont entiers

SI oTmpTxt:nbCols > :nbCols ALORS
    :nbCols = oTmpTxt:nbCols
FIN

Dimension(:Cell,:nbLines+oTmpTxt:nbLines,:nbCols)

POUR L=oTmpTxt:nbLines A 1 PAS -1 //boucle inversée pour optimisation de la boucle (A)
    POUR C=oTmpTxt:nbCols A 1 PAS -1
        :Cell[:nbLines+L,C] = oTmpTxt:Cell[L,C]
    FIN
FIN

:nbLines += oTmpTxt:nbLines
```

```
SI :nbCols > oTmpTxt:nbCols ALORS
    :FillCols(oTmpTxt:nbCols+1,:nbCols,EOT)
FIN
```

Méthode FiltreByCol

```
// Résumé : Supprime toutes les lignes dont la valeur de la colonne <nCol> ne correspond pas à
<sValFiltre>

// Syntaxe :
//[ <Résultat> = ] FiltreByCol (<_sValFiltre> est chaîne [, <_nCol> est entier [, <_Options> est
entier]])

//
// Paramètres :
//  _sValFiltre (chaîne) : Valeur de la colonne à filtrer
//  _nCol (entier - valeur par défaut=1) : Colonne de la valeur à filtrer
//  _Options (entier - valeur par défaut=0) : SansCasse/MotComplet
// Valeur de retour :
//  entier : //     nombre de lignes du tableau
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE FiltreByCol(_sValFiltre est chaîne,_nCol est entier=1,_Options est entier=0)

:DeleteOptBegin()

L est entier
bDel est un booléen
nTotLig est entier=:nbLines

POUR L=1 A nTotLig

    bDel = Vrai
    SELON _Options
        CAS 0
            //Recherche Stricte
            SI :Cell[L,_nCol] = _sValFiltre ALORS
                bDel = Faux
            FIN
        CAS SansCasse
            //Recherche sans attention aux Min/Maj et Espaces
            SI :Cell[L,_nCol] ~= _sValFiltre ALORS
                bDel = Faux
            FIN
        AUTRES CAS
            //MotComplet/DepuisDebut/DepuisFin (ou combinaisons)
            SI Position(:Cell[L,_nCol],_sValFiltre,1,_Options) > 0 ALORS
                bDel = Faux
            FIN
    FIN

    SI bDel ALORS
        :Del(L)
    FIN
FIN

:DeleteOptEnd()

RENVOYER :nbLines
```

Méthode FiltreByLine

```
// Résumé : Supprime toutes les lignes dont la valeur de la ligne ne correspond pas à <sValFiltre>
// Syntaxe :
//[ <Résultat> = ] FiltreByLine (<_sValFiltre> est chaîne [, <_Options> est entier])
//
// Paramètres :
//  _sValFiltre (chaîne) :  Valeur à filtrer
//  _Options (entier - valeur par défaut=0) : <indiquez ici le rôle de _Options>
// Valeur de retour :
//  entier : //     Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE FiltreByLine(_sValFiltre est chaîne,_Options est entier=0)

:DeleteOptBegin()

L est entier
bDel est un booléen
nTotLig est entier=:nbLines

POUR L=1 A nTotLig
    bDel = Vrai
    SELON _Options
        CAS 0
            //Recherche Stricte
            SI :Line(L) = _sValFiltre ALORS
                bDel = Faux
            FIN
        CAS SansCasse
            //Recherche sans attention aux Min/Maj et Espaces
            SI :Line(L) ~= _sValFiltre ALORS
                bDel = Faux
            FIN
        AUTRES CAS
            //MotComplet/DepuisDebut/DepuisFin (ou combinaisons)
            SI Position(:Line(L),_sValFiltre,1,_Options) > 0 ALORS
                bDel = Faux
            FIN
    FIN

    SI bDel ALORS
        :Del(L)
    FIN

FIN

:DeleteOptEnd()

RENVOYER :nbLines
```

Méthode SetKeyValue

```
FONCTION PUBLIQUE SetKeyValue(_sKey est chaîne,LOCAL _sValue est chaîne,_nCol est entier=2,_nOptions est
 entier=0,_bDelIfEmpty est booléen=Faux)

L est entier = :FindInCol(_sKey,1,_nOptions)

SI L > 0 ALORS
    SI _bDelIfEmpty ET _sValue="" ALORS
        :Del(L)
    SINON
        :Cell[L,_nCol] = _sValue
        :ExpandCol(L,_nCol)
    FIN
SINON
    SI PAS _bDelIfEmpty ALORS
        :Add(_sKey+Répète(:sSeparatorX,_nCol-1)+_sValue)
    FIN
FIN

RENVOYER Vrai
```

Méthode ExpandCol

```
PROCEDURE PUBLIQUE ExpandCol(_Line est entier,_Col est entier)

// Si la cellule contient un SeparatorX, il l'éclate sur les cellules de droite

// Si deja dans la derniere col. ou plus on n'eclate pas plus
SI _Col >= :nFixedColumns ALORS RETOUR

C, nColCol, nTotCol sont entiers
sCell est chaîne = :Cell[_Line,_Col]

nColCol = ChaîneOccurrence(sCell,:sSeparatorX) + 1

SI nColCol = 1 ALORS RETOUR

nTotCol = _Col-1+nColCol

SI nTotCol > :nbCols ALORS
    Dimension(:Cell,:nbLines,nTotCol)
    Dimension(:Head,:nbCols)
    // On remplit de EOT l(a/es) nouvelle(s) colonne(s) ajoutée(s)
    :FillCols(:nbCols+1,nTotCol,EOT)
    :nbCols = nTotCol
FIN

POUR C=nTotCol _A_ _Col PAS -1
    :Cell[_Line,C] = ExtraitChaîne(sCell,C,:sSeparatorX)
FIN
```

Méthode FillCol

```
PROCEDURE PUBLIQUE FillCol(_nCol est entier,_Text est chaîne,_nLineMin est entier=1,_nLineMax est entier
=:nbLines)

L est entier

POUR L=_nLineMin _A_ _nLineMax
    :Cell[L,_nCol] = _Text
FIN
```

Méthode FillCols

```
PROCEDURE PUBLIQUE FillCols(_nColMin est entier=1,LOCAL _nColMax est entier=:nbCols,_Text="",_nLineMin
est entier=1,_nLineMax est entier=:nbLines)

C est entier

POUR C=_nColMin A _nColMax
    :FillCol(C,_Text,_nLineMin,_nLineMax)
FIN
```

Méthode TraceTxt

```
// Résumé : Affiche le contenu du tableau dans une Trace Windev, avec colonnage
// Syntaxe :
//[ <Résultat> = ] TraceTxt ( [<bAvecMemo> est booléen [, <nLineMin> est entier [, <nLineMax> est entier
[, <nColMin> est entier [, <nColMax> est entier]]]]])

//
// Paramètres :
//  bAvecMemo (booléen - valeur par défaut=0) : <indiquez ici le rôle de bAvecMemo>
//  nLineMin (entier - valeur par défaut=1) : <indiquez ici le rôle de nLineMin>
//  nLineMax (entier) : <indiquez ici le rôle de nLineMax>
//  nColMin (entier - valeur par défaut=1) : <indiquez ici le rôle de nColMin>
//  nColMax (entier) : <indiquez ici le rôle de nColMax>
// Valeur de retour :
//  Type indéterminé :    <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE TraceTxt(bAvecMemo est booléen=Faux,nLineMin est entier=1,nLineMax est entier=:
nbLines,nColMin est entier=1,LOCAL nColMax est entier=:nbCols)
```

```
// Affiche le contenu dans une Trace tabulée

SI :nbLines = 0 ALORS RENVOYER Faux

tTailleCols est un tableau de :nbCols entiers
nTotLen, nTotLig est un entier
L,C est entier
sL est chaîne
sM,sMemo est chaîne
bMemo est booléen

SI :nbEnteteLines>0 ALORS
    Trace(Remplace(:GetEntete(),:sSeparatorX,"|"))
FIN

SI nLineMin=0 ALORS
    nLineMin = 1
FIN
SI nLineMax=0 ALORS
    nLineMax = :nbLines
FIN

POUR C=nColMin _A_ nColMax
    tTailleCols[C] = :GetMaxColLen(C,Vrai,nLineMin,nLineMax)
    nTotLen+=tTailleCols[C]
FIN

Trace(Répète("-",nTotLen+:nbCols+1))

nTotLig=Min(nLineMax,:nbLines)
POUR L=Min(nLineMin,:nbLines) _A_ nTotLig
    sL = ">"
    sM = "|"

    bMemo = Faux

    POUR C=nColMin _A_ nColMax
        SI :bUseMemo ET bAvecMemo ALORS
            sMemo = :MemoGet(L,C)
        FIN
        SI :Cell[L,C] = EOT ALORS
            sL += Répète("#",tTailleCols[C])+"|"
        SINON
            SI PAS :bCSVQuotes ALORS
                sL += Complète(:Cell[L,C],tTailleCols[C])+"|"
            SINON
                sL += Complète(Remplace(Remplace(:Cell[L,C],RC,"¤"),TAB,"§"),tTailleCols[C])+"|"
            FIN
        FIN

        SI :bUseMemo ET bAvecMemo ALORS
            SI sMemo = "" ALORS
                sM += Répète("#",tTailleCols[C])+"|"
            SINON
                bMemo = Vrai
                sM += Complète(Gauche(sMemo,tTailleCols[C]),tTailleCols[C])+"|"
            FIN
        FIN

    FIN

    SI :bUseMemo ET bAvecMemo ALORS sL += "// "+:MemoGet(L)
    Trace(sL)
    SI :bUseMemo ET bAvecMemo ET bMemo ALORS Trace(sM)
FIN

Trace(Répète("-",nTotLen+:nbCols+1))

RENVOYER :nbLines
```

Méthode GetMaxColLen

```
FONCTION GetMaxColLen(_Col est entier,_bForTracing est booléen=Faux,LOCAL _nLineMin est entier=1,LOCAL
_nLineMax est entier=:nbLines)

L, nRes est entier

SI _bForTracing ALORS
    POUR L=_nLineMin A _nLineMax
        nRes = Max(nRes,Taille(Remplace(:Cell[L,_Col],EOT,"<EOT>")))
    FIN
SINON
    POUR L=_nLineMin A _nLineMax
        nRes = Max(nRes,Taille(:Cell[L,_Col]))
    FIN
FIN

RENVOYER nRes
```

Méthode TxtVersZMem

```
PROCEDURE PUBLIQUE TxtVersZMem(_ZMem,_nColCle est entier=1,_SeparatorX=:sSeparatorX, bClearZMem=Faux)

:JaugeMinMax(0,:nbLines)

L, C, nSepLen sont entier
sValRetour sont chaînes
nSepLen = Taille(_SeparatorX)
bResult est entier = Vrai

SI bClearZMem ALORS
    MemCrée(_ZMem)
FIN

POUR L=1 _A_ :nbLines

    sValRetour = ""
    POUR C=1 _A_ :nbCols
        SI C <> _nColCle sValRetour += :Cell[L,C]+_SeparatorX
    FIN
    SI sValRetour <> "" ALORS
        sValRetour = Gauche(sValRetour,Taille(sValRetour)-nSepLen)
    FIN
    bResult = bResult ET MemAjoute(_ZMem,:Cell[L,_nColCle],sValRetour)

    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)

FIN

:JaugeSet(0)

RENVOYER bResult
```

Méthode zMemVersTxt

```
PROCEDURE PUBLIQUE zMemVersTxt(_ZMem, bCleZMem=Vrai, bValRetourZMem=Vrai, bClearTab=Faux)

:JaugeMinMax(0,MemOccurrence(_ZMem))

// Transfere le contenu d'une zone mémoire
L est entier
S est chaîne

SI PAS bClearTab ALORS

    // On ajoute au tableau

    MemPremier(_ZMem)
    TANTQUE PAS MemEnDehors(_ZMem)
        L++
        MemPositionne(_ZMem,L)
        S = ""
        SI bCleZMem ALORS
            S += MemValClé(_ZMem)
```

```
            SI bValRetourZMem ALORS
                S += :sSeparatorX
            FIN
        FIN
        SI bValRetourZMem ALORS
            S += MemRecupère(_ZMem,L)
        FIN
        :Add(S)
        MemSuivant(_ZMem)

        SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
    FIN

SINON

    // On crée le tableau

    :Reset()

    :nbLines = MemOccurrence(_ZMem)
    Dimension(:Cell,:nbLines,2)

    MemPremier(_ZMem)
    TANTQUE PAS MemEnDehors(_ZMem)
        L++
        MemPositionne(_ZMem,L)

        S = ""
        SI bCleZMem ALORS
            S += MemValClé(_ZMem)
            SI bValRetourZMem ALORS
                S += :sSeparatorX
            FIN
        FIN
        SI bValRetourZMem ALORS
            S += MemRecupère(_ZMem,L)
        FIN

        :Cell[L,1] = S
        MemSuivant(_ZMem)

        SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
    FIN

    :ExtractCols()

FIN

:JaugeSet(0)
```

Méthode Clear

```
PROCEDURE PUBLIQUE Clear(_bHeadToo=Faux)

// Vide le tableau mais garde le nb de colonnes et l'entete

:nbCols=Max(:nbCols,1)
:nbLines=0

QUAND EXCEPTION DANS
    TableauSupprimeTout(:Cell)
FAIRE
    ExceptionActive()
FIN
//SI TableauInfo(:Cell,tiNombreLignes) > 0 ALORS
//  Dimension(:Cell,1,:nbCols)
//  :SetLine(1,"")
//fin

QUAND EXCEPTION DANS
    Dimension(:Memo,0)
FAIRE
    ExceptionActive()
FIN
```

```
:nMemoCount = 0
:bUseMemo   = Faux

:nCRCInput  = 0
:sID_Input  = ""

SI _bHeadToo ALORS
    :Head[1]=""
    :ExtractEntete()
FIN
```

Méthode MemAjoute

```
PROCEDURE PUBLIQUE MemAjoute(sZoneMem,sValAjoutee,sValRetour)

// Pour compatibilité avec MemAjoute

// Ex: MemAjoute("truc","ValAjoutee","ValRetour") << Nom zone mem ignoré
:Add(sValAjoutee+:sSeparatorX+sValRetour)
:nMemEnCours = :nbLines

//pour desactivation warning variable non utilisée
SI sZoneMem="" ALORS
FIN
```

Méthode MemModifie

```
FONCTION PUBLIQUE MemModifie(sZoneMem,sValAjoutee,sValRetour,nIndice est entier)

// Pour compatibilité avec MemModifie

// Ex: MemModifie("truc","ValAjoutee","ValRetour") << Nom zone mem ignoré
:nMemEnCours = nIndice
:SetLine(nIndice,sValAjoutee+:sSeparatorX+sValRetour)

RENVOYER :MemPositionne(sZoneMem,nIndice)
```

Méthode MemSupprime

```
PROCEDURE PUBLIQUE MemSupprime(sZoneMem,nIndice est entier)

// Pour compatibilité avec MemSupprime

// Ex: MemModifie("truc","ValAjoutee","ValRetour") << Nom zone mem ignoré
:nMemEnCours = nIndice-1
:Del(nIndice)

//pour desactivation warning variable non utilisée
SI sZoneMem="" ALORS
FIN
```

Méthode MemRecupère

```
FONCTION PUBLIQUE MemRecupère(sZoneMem,nIndice est entier)

// Pour compatibilité avec MemRecupère

sRes est chaîne
C est entier

POUR C=2 _A_ :nbCols
    SI :Cell[nIndice,C] = EOT ALORS SORTIR
    sRes += :Cell[nIndice,C]+:sSeparatorX
FIN

SI sRes <> "" ALORS
    sRes = Gauche(sRes,Taille(sRes)-Taille(:sSeparatorX))
FIN

//pour desactivation warning variable non utilisée
SI sZoneMem="" ALORS
FIN
```

```
RENVOYER sRes
```

Méthode MemValClé

```
FONCTION PUBLIQUE MemValClé(*)

// Pour compatibilité avec MemValClé

SI :nMemEnCours > :nbLines ALORS
    RENVOYER ""
FIN

RENVOYER :Cell[:nMemEnCours,1]
```

Méthode MemOccurrence

```
FONCTION PUBLIQUE MemOccurrence(*)

// Pour compatibilité avec MemOccurrence

RENVOYER :nbLines
```

Méthode MemPositionne

```
FONCTION PUBLIQUE MemPositionne(sZoneMem,nIndice est entier)

// Pour compatibilité avec MemPositionne

SI nIndice <= :nbLines ALORS
    :nMemEnCours = nIndice
    RENVOYER :MemRecupère(sZoneMem,nIndice)
SINON
    RENVOYER ""
FIN
```

Méthode MemEnCours

```
FONCTION PUBLIQUE MemEnCours(sZoneMem)

// Pour compatibilité avec MemEnCours

//pour desactivation warning variable non utilisée
SI sZoneMem="" ALORS
FIN

RENVOYER :nMemEnCours
```

Méthode MemSuivant

```
FONCTION PUBLIQUE MemSuivant(sZoneMem)

// Pour compatibilité avec MemSuivant

:bMemEnDehors = Faux

SI :nMemEnCours < :nbLines ALORS
    :nMemEnCours++
    RENVOYER :MemRecupère(sZoneMem,:nMemEnCours)
SINON
    :bMemEnDehors = Vrai
    RENVOYER ""
FIN
```

Méthode MemPrécédent

```
FONCTION PUBLIQUE MemPrécédent(sZoneMem)

// Pour compatibilité avec MemPrécédent

:bMemEnDehors = Faux

SI :nMemEnCours > 1 ALORS
    :nMemEnCours--
    RENVOYER :MemRecupère(sZoneMem,:nMemEnCours)
SINON
    :bMemEnDehors = Vrai
    RENVOYER ""
FIN
```

Méthode MemRecherche

```
FONCTION PUBLIQUE MemRecherche(sZoneMem,sValRech,bElement=Vrai)

// Pour compatibilité avec MemRecherche

:bMemTrouve   = Faux
:bMemEnDehors = Faux

SELON bElement
    CAS Vrai
        :nMemEnCours = :FindInCol(sValRech,1)
        SI :nMemEnCours = 0 ALORS
            :bMemEnDehors = Vrai
            RENVOYER ""
        SINON
            :bMemTrouve = Vrai
//          RENVOYER :MemValClé(sZoneMem)
            RENVOYER :MemPositionne(sZoneMem,:nMemEnCours)
        FIN

    CAS Faux
        :nMemEnCours = :FindInMultiCol(sValRech,2)
        SI :nMemEnCours = 0 ALORS
            :bMemEnDehors = Vrai
            RENVOYER -1
        SINON
            :bMemTrouve   = Vrai
            RENVOYER :nMemEnCours
        FIN
FIN

RENVOYER Faux
```

Méthode MemTrie

```
PROCEDURE PUBLIQUE MemTrie(sZoneMem,bSens=Vrai,bElement=Vrai)

SI bElement ALORS
    :Sort(1,bSens)
SINON
    C est entier
    POUR C=:nbCols A 2 PAS -1
        :Sort(C,bSens)
    FIN
FIN

//pour desactivation warning variable non utilisée
SI sZoneMem="" ALORS
FIN
```

Méthode FindInMultiCol

```
FONCTION PUBLIQUE FindInMultiCol(_sRech,_nColDeb est entier=1,LOCAL _nColFin est entier=0,_nLineMin est entier=1,LOCAL _nLineMax est entier=0)

// Recherche dans un bloc de colonnes

SI _nLineMax=0 ALORS
    _nLineMax=:nbLines
FIN

SI _nColFin=0 ALORS
    _nColFin=:nbCols
FIN

sBloc est chaîne
L,C est entier

POUR L=_nLineMin _A_ :nbLines

    POUR C=_nColDeb A _nColFin
        SI :Cell[L,C] = EOT ALORS SORTIR
        sbloc += :Cell[L,C]+:sSeparatorX
    FIN

    SI sBloc <> "" ALORS
        sbloc = Gauche(sbloc,Taille(sbloc)-Taille(:sSeparatorX))
    FIN

    SI sBloc = _sRech ALORS
        RENVOYER L
    FIN

FIN

RENVOYER 0
```

Méthode Premier

```
// Résumé : Positionne le curseur :EnCours sur la première ligne, et renvoie le contenu de la colonne spéficifiée ou la ligne sinon

// Syntaxe :
//[ <Résultat> = ] Premier ( [<_nCol> est entier])
//
// Paramètres :
//  _nCol (entier - valeur par défaut=0) : <indiquez ici le rôle de nCol>
// Valeur de retour :
//  chaîne :   renvoie le contenu de la colonne <nCol> spéficifiée (ou la ligne par défaut)
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE Premier(_nCol est entier=0)

sRes est chaîne=EOT

SI :nbLines>0 ET :nbCols>=_nCol ALORS
    SI _nCol=0 ALORS
        sRes = :Line(1)
    SINON
        sRes = :Cell[1,_nCol]
    FIN
FIN

SI :nbLines<=1 _ET_ sRes _DANS_ ("",EOT) ALORS
    :EnDehors = Vrai
    :EnCours = 0
SINON
    :EnDehors = Faux
    :EnCours = 1
FIN

RENVOYER sRes
```

Méthode Suivant

```
// Résumé : Positionne le curseur :EnCours sur la ligne suivante, et renvoie le contenu de la colonne
spéficifiée ou la ligne sinon

// Syntaxe :
// [ <Résultat> = ] Suivant ( [<nCol> est entier])
//
// Paramètres :
//  nCol (entier - valeur par défaut=0) : <indiquez ici le rôle de nCol>
// Valeur de retour :
//  Type indéterminé : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Suivant(nCol est entier=0)

SI :EnCours < :nbLines ALORS
    :EnCours++
    :EnDehors = Faux
SINON
    :EnDehors = Vrai
    RENVOYER ""
FIN

SI nCol=0 ALORS
    RENVOYER :Line(:EnCours)
SINON
    RENVOYER :Cell[:EnCours,nCol]
FIN
```

Méthode Precedent

```
// Résumé : Positionne le curseur :EnCours sur la ligne Precedente, et renvoie le contenu de la colonne
spéficifiée ou la ligne sinon

// Syntaxe :
//[ <Résultat> = ] Precedent ( [<nCol> est entier])
//
// Paramètres :
//  nCol (entier - valeur par défaut=0) : < indiquez ici le rôle de nCol >
// Valeur de retour :
//  Type indéterminé :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Precedent(nCol est entier=0)

SI :EnCours > 1 ALORS
    :EnCours--
    :EnDehors = Faux
SINON
    :EnDehors = Vrai
    RENVOYER ""
FIN

SI nCol=0 ALORS
    RENVOYER :Line(:EnCours)
SINON
    RENVOYER :Cell[:EnCours,nCol]
FIN
```

Méthode Dernier

```
// Résumé : Positionne le curseur :EnCours sur la derniere ligne, et renvoie le contenu de la colonne
spéficifiée ou la ligne sinon

// Syntaxe :
//[ <Résultat> = ] Dernier ( [<nCol> est entier])
//
// Paramètres :
//  nCol (entier - valeur par défaut=0) : < indiquez ici le rôle de nCol >
// Valeur de retour :
//  Type indéterminé :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Dernier(nCol est entier=0)

SI :nbLines=0 ALORS
    :EnDehors = Vrai
    :EnCours = 0
    RENVOYER ""
FIN

:EnDehors = Faux
:EnCours = :nbLines

SI nCol=0 ALORS
    RENVOYER :Line(:EnCours)
SINON
    RENVOYER :Cell[:EnCours,nCol]
FIN
```

Méthode TableVersTxt

```
// Résumé : Transfert(en ajout) d'une Table "Graphique" Mémoire vers la classe
// Syntaxe :
//[ <Résultat> = ] TableVersTxt (<_TABLE> [, <bHeadToo> est booléen [, <bUseLibCol> est booléen]])
//
// Paramètres :
//  _TABLE : Champ Table
//  bHeadToo (booléen - valeur par défaut=0) : <indiquez ici le rôle de bHeadToo>
//  bUseLibCol (booléen - valeur par défaut=0) : Nom(faux) ou Libellé(vrai) du titre de la colonne
// Valeur de retour :
//  Type indéterminé :        //    Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE TableVersTxt(_TABLE,bHeadToo est booléen=Faux,bUseLibCol est booléen=Faux)
```

```
RENVOYER Faux
```

```
L,C sont entiers
nTotLig est entier=TableOccurrence(_TABLE)

POUR L=1 A nTotLig
    :Add(Remplace(_TABLE[L],TAB,:sSeparatorX))
FIN

C = TableOccurrence(_TABLE,toColonne)
SI C > :nbCols ALORS
    Dimension(:Cell,:nbLines,C)
    Dimension(:Head,C)
    :FillCols(:nbCols+1,C)
    :nbCols = C
FIN

SI bHeadToo ALORS
    POUR L=1 A C
```

```
        SI bUseLibCol ALORS
            :Head[L] = {TableEnumèreColonne(_TABLE,L),indChamp}..Libellé
        SINON
            :Head[L] = TableEnumèreColonne(_TABLE,L)
        FIN
    FIN
FIN

RENVOYER nTotLig
```

Méthode TxtVersTable

```
// Résumé : Transfert(en ajout) de la classe vers une Table "graphique" de type mémoire
// Syntaxe :
//[ <Résultat> = ] TxtVersTable (<_TABLE> [, <bHeadToo> est booléen [, <nDecaleCol> est entier [,
<sPrefix> est chaîne [, <sSuffix> est chaîne]]]])

//
// Paramètres :
//  _TABLE : <indiquez ici le rôle de _TABLE>
//
bHeadToo (booléen - valeur par défaut=0) : Affecte les titres de colonnes (libellé) avec l'entete interne
//  nDecaleCol (entier - valeur par défaut=0) : <indiquez ici le rôle de nDecaleCol>
//  sPrefix (chaîne) : <indiquez ici le rôle de sPrefix>
//  sSuffix (chaîne) : <indiquez ici le rôle de sSuffixe>
// Valeur de retour :
//  booléen :            //     Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE TxtVersTable(_TABLE,bHeadToo est booléen=Faux,nDecaleCol est entier=0,sPrefix est
chaîne="",sSuffix est chaîne="")

L est entier
nTotLig est entier=:nbLines
nTotCol est entier=:nbCols
sNomCol est une chaîne
sDecal est chaîne
sLigne est chaîne

SI nDecaleCol >= 0 ALORS
    SI sPrefix="" ALORS
        sDecal = Répète(TAB,nDecaleCol)
    FIN
SINON
    RENVOYER Faux
FIN

POUR L=1 A nTotLig
    sLigne = sDecal+Remplace(:Line(L),:sSeparatorX,TAB)
    SI PAS WL.TableAjoute(_TABLE,sPrefix+sLigne+sSuffix) ALORS
        RENVOYER Faux
    FIN
FIN

SI bHeadToo ALORS
    //Construction de la table
    POUR L=1 A nTotCol
        sNomCol=TableEnumèreColonne(_TABLE,L+nDecaleCol)
        SI sNomCol<>"" ALORS
            {sNomCol,indChamp}..Libellé = :Head[L]
        FIN
    FIN
FIN

RENVOYER Vrai
```

Méthode AddCol

```
// Résumé : Ajoute une colonne à la fin
// Syntaxe :
//[ <Résultat> = ] AddCol ( [<sFill> est chaîne])
//
// Paramètres :
//  sFill (chaîne - valeur par défaut=EOT) : Valeur affectée à la nouvelle colonne
// Valeur de retour :
//  entier :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE AddCol(sFill est chaîne=EOT)

:Add(Répète(:sSeparatorX,:nbCols)+sFill)
:Del(:nbLines)

RENVOYER :nbCols
```

Méthode Resize

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//Resize (<_Lines> est entier, <_Cols> est entier)
//
// Paramètres :
//  _Lines (entier) : <indiquez ici le rôle de _Lines>
//  _Cols (entier) : <indiquez ici le rôle de _Cols>
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE Resize(LOCAL _Lines est entier,LOCAL _Cols est entier)

C est entier
SI _Cols > :nbCols ALORS
    POUR C=:nbCols+1 A _Cols
        :AddCol()
    FIN
SINON SI _Cols < :nbCols ALORS
    Dimension(:Cell,:nbLines,_Cols)
    Dimension(:Head,_Cols)
    :nbCols=_Cols
FIN

L est entier
SI _Lines > :nbLines ALORS
    POUR L=:nbLines+1 A _Lines
        :Add(EOT)
    FIN
SINON SI _Lines < :nbLines ALORS
    Dimension(:Cell,_Lines,:nbCols)
    :nbLines = _Lines
FIN
```

Méthode IniVersTxt

```
// Résumé : Charge un fichier INI dans le tableau dynamique
// Syntaxe :
//[ <Résultat> = ] IniVersTxt (<_FichierIni> est chaîne [, <_Section> est chaîne [, <_bMerge> est
booléen]])

//
// Paramètres :
//  _FichierIni (chaîne) : Chemin du fichier INI
//  _Section (chaîne) : <indiquez ici le rôle de _Section>
//  _bMerge (booléen - valeur par défaut=0) : <indiquez ici le rôle de _bMerge>
// Valeur de retour :
//
booléen :                                <indiquez ici les valeurs possibles ainsi que leur
interprétation>
```

```
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION IniVersTxt(_FichierIni est chaîne,_Section est chaîne="",_bMerge est booléen=Faux)

uSizeStruct est entier=fTaille(_FichierIni) //TROP LENT en WD15 ?

//id est entier = fOuvre(_FichierIni, foLecture)
//SI id <> -1 ALORS
//  // Récupération de la taille
//  uSizeStruct = fPositionne(id, 0, fpFin)
//  fFerme(id)
//FIN

//Optimisation via CRC
SI _bMerge ALORS
    //On désactive l'optimisation
    :bCRCInput = Faux
FIN
SI :bCRCInput ALORS
    nCRC est entier sans signe
    nCRC = sCalculeCrc32(fChargeTexte(_FichierIni))
    SI nCRC _ET_ :nCRCInput = nCRC ALORS
        RENVOYER Vrai
    FIN
    SI :nbLines ALORS :Clear() //(SI PAS _bMerge)
    :nCRCInput = nCRC
FIN

:sID_Input = _FichierIni

SI uSizeStruct<=0 ALORS RENVOYER Faux

:nFixedColumns = 3

SI :bLinux ALORS
    //Linux - IniLit("","","") ne fonctionne pas WD12 55n
    RENVOYER :IniVersTxt_Linux(_FichierIni,_Section,_bMerge)
FIN

cTxtSections est un cTxt
cTxtSections:nModeMultitache=:nModeMultitache

SI _Section = "" ALORS
    cTxtSections:Parse(IniLit("", "", "", _FichierIni))
SINON
    cTxtSections:Add(_Section)
FIN

nS,nRead est entier
sContent,sSection,sLine,sKey sont chaîne

POUR nS=1 _A_ cTxtSections:nbLines
    sContent=Complète(sContent,uSizeStruct)
    sSection=cTxtSections:Cell[nS,1]
    nRead = API("KERNEL32","GetPrivateProfileSectionA",sSection,&sContent,uSizeStruct,_FichierIni)
    SI nRead > 0 ALORS
        sSection=Majuscule(cTxtSections:Cell[nS,1])
        sContent=SansEspace(sContent[[A nRead]])
        POUR TOUTE CHAINE sLine DE sContent SEPAREE PAR Caract(0)
            SI sLine="" CONTINUER
            sKey=ExtraitChaîne(sLine,1,"=")
            SI PAS _bMerge ALORS
                :Add(sSection+:sSeparatorX+SansEspace(sKey)+:sSeparatorX+SansEspace(sLine[[Taille(sKey)+2
                 A]]))
            SINON
                :IniEcrit(sSection,SansEspace(sKey),SansEspace(sLine[[Taille(sKey)+2 A]]))
            FIN
        FIN
    FIN
FIN

RENVOYER Vrai
```

Méthode TxtVersIni

```
// Résumé : Crée un fichier ini
// Syntaxe :
//[ <Résultat> = ] TxtVersIni (<_FichierIni> est chaîne [, <_Section> est chaîne [, <_bReset> est
booléen]])

//
// Paramètres :
//  _FichierIni (chaîne) : <indiquez ici le rôle de _FichierIni>
//  _Section (chaîne) : <indiquez ici le rôle de _Section>
//  _bReset (booléen - valeur par défaut=0) : <indiquez ici le rôle de _bReset>
// Valeur de retour :
//  booléen :   <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE TxtVersIni(_FichierIni est chaîne,_Section est chaîne="",_bReset est booléen=Faux)

SI _bReset ALORS
    SI _Section = "" ALORS
        //On vide le fichier INI avant
        fSupprime(_FichierIni,frLectureSeule)
    FIN
FIN

SI :nbCols < 3 ALORS
    RENVOYER Faux
FIN

L est entier
sOldSection est chaîne

POUR L=1 _A_ :nbLines
    SI _Section = "" OU _Section=:Cell[L,1] ALORS

        SI sOldSection <> :Cell[L,1] ET _bReset ALORS
            //On vide le fichier INI avant
            WL.INIEcrit(:Cell[L,1],"","",_FichierIni)
            sOldSection = :Cell[L,1]
        FIN

        WL.INIEcrit(:Cell[L,1],:Cell[L,2],:Line(L,3),_FichierIni)
    FIN
FIN

RENVOYER Vrai
```

Méthode RegVersTxt

```
PROCEDURE RegVersTxt(LOCAL _sRootKey est chaîne,_nbLevels est entier=-1,_bTypesInComment est booléen=Vrai
)

//A FAIRE:Gérer le nombre de niveaux à recuperer
SI _nbLevels>0 ALORS
FIN
```

```
RENVOYER Faux
```

```
SI PAS RegistreExiste(_sRootKey) ALORS
    RENVOYER Faux
FIN

nLev est entier = 1
sKey est chaîne
nVal est entier
bRead,bToRead est booléen
sKeyName est chaîne
```

```
hKey est un entier // Type C :HKEY
sSubKey est chaîne

sValueName est chaîne ASCIIZ sur 256
BufferSize est entier = 256

dwType est un entier sans signe

REG_NONE                       est entier = 0
REG_SZ                         est entier = 1
REG_EXPAND_SZ                  est entier = 2
REG_BINARY                     est entier = 3
REG_DWORD                      est entier = 4
REG_DWORD_BIG_ENDIAN           est entier = 5
REG_LINK                       est entier = 6
REG_MULTI_SZ                   est entier = 7
REG_RESOURCE_LIST              est entier = 8
REG_FULL_RESOURCE_DESCRIPTOR   est entier = 9
REG_RESOURCE_REQUIREMENTS_LIST est entier = 10

HKEY_CLASSES_ROOT     est un entier sans signe = 0x80000000
HKEY_CURRENT_USER     est un entier sans signe = 0x80000001
HKEY_LOCAL_MACHINE    est un entier sans signe = 0x80000002
HKEY_USERS            est un entier sans signe = 0x80000003
HKEY_PERFORMANCE_DATA est un entier sans signe = 0x80000004
HKEY_CURRENT_CONFIG   est un entier sans signe = 0x80000005
HKEY_DYN_DATA         est un entier sans signe = 0x80000006

KEY_QUERY_VALUE est entier = 1

HKEY_ROOT est un entier sans signe

sSubKey = ExtraitChaîne(_sRootKey,1,"\")
SELON sSubKey
    CAS "HKEY_CLASSES_ROOT"     : HKEY_ROOT = HKEY_CLASSES_ROOT
    CAS "HKEY_CURRENT_USER"     : HKEY_ROOT = HKEY_CURRENT_USER
    CAS "HKEY_LOCAL_MACHINE"    : HKEY_ROOT = HKEY_LOCAL_MACHINE
    CAS "HKEY_USERS"            : HKEY_ROOT = HKEY_USERS
    CAS "HKEY_PERFORMANCE_DATA" : HKEY_ROOT = HKEY_PERFORMANCE_DATA
    CAS "HKEY_CURRENT_CONFIG"   : HKEY_ROOT = HKEY_CURRENT_CONFIG
    CAS "HKEY_DYN_DATA"         : HKEY_ROOT = HKEY_DYN_DATA
FIN

sPath    est chaîne = _sRootKey
sPathSep est chaîne

nOldLev est entier

RECURSIVITE:

TANTQUE sKey <> "" OU nVal=0

    sSubKey=Remplace(sPath,ExtraitChaîne(_sRootKey,1,"\")+"\","")

    hKey=0
    SI API("ADVAPI32.DLL","RegOpenKeyExA",HKEY_ROOT,sSubKey,0,KEY_QUERY_VALUE,&hKey) = 0 ALORS

        nVal=0; bRead=Vrai; bToRead=Vrai
        TANTQUE bToRead ET bRead
            BufferSize = 256
            bToRead = PAS API("ADVAPI32.DLL","RegEnumValueA",hKey,nVal,&sValueName,&BufferSize,Null,&
            dwType,Null,Null)
            nVal++
            SI PAS bToRead ALORS
                //Valeur par défaut de la clé
                sValueName=""
                dwType=1
                :Add(sPathSep+"\"+:sSeparatorX+RegistreLit(sPath,"",bRead))
            SINON
                :Add(sPathSep+sValueName+:sSeparatorX+RegistreLit(sPath,sValueName,bRead))
            FIN
            SI _bTypesInComment ALORS
                SELON dwType
                    CAS REG_SZ
```

```
                    :MemoSet(:nbLines,0,"REG_SZ"+TAB+dwType)
                CAS REG_EXPAND_SZ
                    :MemoSet(:nbLines,0,"REG_EXPAND_SZ"+TAB+dwType)
                CAS REG_BINARY
                    :MemoSet(:nbLines,0,"REG_BINARY"+TAB+dwType)
                CAS REG_DWORD
                    :MemoSet(:nbLines,0,"REG_DWORD"+TAB+dwType)
                CAS REG_DWORD_BIG_ENDIAN
                    :MemoSet(:nbLines,0,"REG_DWORD_BIG_ENDIAN"+TAB+dwType)
                CAS REG_LINK
                    :MemoSet(:nbLines,0,"REG_LINK"+TAB+dwType)
                CAS REG_MULTI_SZ
                    :MemoSet(:nbLines,0,"REG_MULTI_SZ"+TAB+dwType)
                    STOP
                CAS REG_RESOURCE_LIST
                    :MemoSet(:nbLines,0,"REG_RESOURCE_LIST"+TAB+dwType)
                CAS REG_FULL_RESOURCE_DESCRIPTOR
                    :MemoSet(:nbLines,0,"REG_FULL_RESOURCE_DESCRIPTOR"+TAB+dwType)
                CAS REG_RESOURCE_REQUIREMENTS_LIST
                    :MemoSet(:nbLines,0,"REG_RESOURCE_REQUIREMENTS_LIST"+TAB+dwType)
                AUTRES CAS
                    :MemoSet(:nbLines,0,"REG_NONE"+TAB+dwType)
            FIN
        FIN

    FIN

    API("ADVAPI32.DLL","RegCloseKey",hKey)
FIN

SI Position(RegistrePremièreSousClé(sPath),_sRootKey) = 1 ALORS
    sKey = RegistrePremièreSousClé(sPath)
    sKeyName = sKey[[Taille(sPath)+2 A]]
    sPath=sKey
    sPathSep+=sKeyName+:sSeparatorX
    nLev++
    GOTO RECURSIVITE
SINON SI Position(RegistreCléSuivante(sPath),_sRootKey) = 1 ALORS
    // Clé suivante dans le meme niveau
    sKey     = RegistreCléSuivante(sPath)
    sKeyName = sKey[[Position(sKey,"\",Taille(sPath),DepuisFin)+1 A]]
    sPath=sKey
    //sPath=Rem
    SI ChaîneOccurrence(sPathSep,:sSeparatorX) = 1 ALORS
        sPathSep=sKeyName+:sSeparatorX
    SINON
        sPathSep=sPathSep[[A Position(sPathSep,:sSeparatorX,Taille(sPathSep)-Taille(:sSeparatorX)-1,
        DepuisFin)-1]]+:sSeparatorX+sKeyName+:sSeparatorX
    FIN
SINON

    nOldLev = nLev
    TANTQUE Position(sPath, _sRootKey) > 0
        // On remonte d'un cran
        sPath = sPath[[A Taille(sPath)-Taille(sKeyName)]]
        nLev--

        sKey = RegistreCléSuivante(sPath)
        SI sKey <> "" ALORS
            sPath = sKey
        FIN

        SI Droite(sPath,1) = "\" ALORS
            sPath = sPath[[A Taille(sPath)-1]]
        FIN

        SI ChaîneOccurrence(sPathSep,:sSeparatorX) > 1 ALORS
            sPathSep=sPathSep[[A Position(sPathSep,:sSeparatorX,Taille(sPathSep)-Taille(:sSeparatorX)
            -1,DepuisFin)-1]]+:sSeparatorX
        FIN

        sKeyName = sPath[[Position(sPath,"\",Taille(sPath),DepuisFin)+1 A]]

        SI sKey <> "" ALORS
```

```
                SI ChaîneOccurrence(sPathSep,:sSeparatorX) = 1 ALORS
                    sPathSep=sKeyName+:sSeparatorX
                SINON
                    sPathSep=sPathSep[[A Position(sPathSep,:sSeparatorX,Taille(sPathSep)-Taille(:
                    sSeparatorX)-1,DepuisFin)-1]]+:sSeparatorX+sKeyName+:sSeparatorX
                FIN

                SORTIR
            FIN
        FIN

        SI Position(sPath, _sRootKey)=0 ALORS
            sKey=""
        FIN

    FIN

FIN

RENVOYER Vrai
```

Méthode RepVersTxt

```
// Résumé : Charge le listing d'un répertoire dans le tableau
// Syntaxe :
// RepVersTxt (<_sRep> [, <_sNoeud> [, <_nModeRech> [, <_nbLevel> [, <_nModeAffic>]]]])
//
// Paramètres :
//  _sRep : <indiquez ici le rôle de _sRep>
//  _sNoeud : <indiquez ici le rôle de _sNoeud>
//  _nModeRech (valeur par défaut=56) : < indiquez ici le rôle de _nModeRech >
//  _nbLevel (valeur par défaut=0) : < indiquez ici le rôle de _nbLevel >
//  _nModeAffic (valeur par défaut=2) : < indiquez ici le rôle de _nModeAffic >
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE RepVersTxt(LOCAL _sRep,LOCAL _sNoeud="",LOCAL _nModeRech=frRépertoire+frRécursif, LOCAL
_nbLevel=0, LOCAL _nModeAffic=2)

//Mode import :
// 0 : _sNoeud | SubRep1 | SubRep2 | ... | Fichier ou Rep\
// 1 : _sNoeud | _sRep | Fichier ou Rep\
// 2 : _sRep   | Fichier ou Rep\

SI fRep(_sRep,frRépertoire) = "" ALORS RETOUR
fRep("")

R est entier
sRep est chaîne
sLstRep est chaîne
sRoot est chaîne

SI _nModeRech <> frFichier ET _nModeRech <> frRépertoire ALORS
    sRep = fRep(ComplèteRep(_sRep) + "*.*",frRépertoire)
    SI sRep <> "" ALORS
        TANTQUE sRep <> ""
            sLstRep+=sRep+RC
            sRep = fRep("",frRépertoire)
        FIN
        fRep("")
    FIN

    R=1
    sRep = ExtraitChaîne(sLstRep,R,RC)
    TANTQUE sRep <> ""

        // Ajout du Répertoire dans la liste
        SELON _nModeRech
            CAS frFichier
            CAS frFichier+frRécursif
            AUTRES CAS
```

```
                SELON _nModeAffic
                    CAS 0
                        :Add(_sNoeud + :sSeparatorX + sRep+"\")
                    CAS 1
                        sRoot = ExtraitChaîne(_sNoeud,1,:sSeparatorX)
                        :Add(sRoot+ :sSeparatorX +Remplace(_sNoeud[[Taille(sRoot)+Taille(:sSeparatorX)+1
                        A]],:sSeparatorX,"\") + :sSeparatorX + sRep+"\")
                    CAS 2
                        :Add(Remplace(_sNoeud,:sSeparatorX,"\") + :sSeparatorX + sRep+"\")
                FIN
        FIN

        // Recherche récursive
        SI _nModeRech = frFichier+frRécursif OU _nModeRech = frRépertoire+frRécursif OU _nModeRech =
        frRépertoire+frFichier+frRécursif ALORS
            :RepVersTxt(ComplèteRep(_sRep) + sRep, _sNoeud + :sSeparatorX + sRep, _nModeRech, _nbLevel,
            _nModeAffic)
        FIN

        R++
        sRep=ExtraitChaîne(sLstRep,R,RC)
    FIN
FIN

// Ajout des fichiers dans la liste
SI _nModeRech <> frRépertoire ET _nModeRech <> frRécursif+frRépertoire ALORS
    sRep = fRep(ComplèteRep(_sRep) + "*.*",frFichier)
    SI sRep <> "" ALORS
        TANTQUE sRep <> ""
            SELON _nModeAffic
                CAS 0
                    :Add(_sNoeud + :sSeparatorX + sRep)
                CAS 1
                    sRoot = ExtraitChaîne(_sNoeud,1,:sSeparatorX)
                    :Add(sRoot+ :sSeparatorX +Remplace(_sNoeud[[Taille(sRoot)+Taille(:sSeparatorX)+1 A]],
                    :sSeparatorX,"\") + :sSeparatorX + sRep)
                CAS 2
                    :Add(Remplace(_sNoeud,:sSeparatorX,"\") + :sSeparatorX + sRep)
            FIN
            sRep = fRep("",frFichier)
        FIN
        fRep("")
    FIN
FIN
```

Méthode Move

```
// Résumé : Coupe et insère une ligne avant l'indice de destination
// Syntaxe :
// Move (<nLnOrig> est entier, <nLnDest> est entier)
//
// Paramètres :
//  nLnOrig (entier) : Indice d'origine
//  nLnDest (entier) : Indice de destination
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Move(nLnOrig est entier, nLnDest est entier)

sLigne est chaîne = :Line(nLnOrig)

SI nLnOrig < nLnDest ALORS
    SI :bUseMemo ALORS
        // A FAIRE: TESTER CETTE POSSIBILITE
        :DeleteOptBegin()
        :Del(nLnOrig)
        :Insert(nLnDest+1,sLigne)
        :MemoMove(nLnOrig,0,nLnDest+1)
        :DeleteOptEnd()
    SINON
        :Del(nLnOrig)
```

```
        :Insert(nLnDest,sLigne)
    FIN
SINON SI nLnOrig > nLnDest ALORS
    :Insert(nLnDest,sLigne)
    SI :bUseMemo ALORS
        :MemoMove(nLnOrig,0,nLnDest)
    FIN
    :Del(nLnOrig+1)
FIN
```

Méthode MemoAdd

```
// Résumé : Affecte un texte mémo à une cellule ou ligne (si C=0)
// Attention, ne remplace pas le memo associé (a utiliser apres MemoDel), utilisez MemoSet si mémo unique
// Syntaxe :
// MemoAdd (<L> est entier, <C> est entier, <sData> est chaîne)
//
// Paramètres :
//  L (entier) : <indiquez ici le rôle de L>
//  C (entier) : <indiquez ici le rôle de C>
//  sData (chaîne) : <indiquez ici le rôle de sData>
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE MemoAdd(L est entier,C est entier,sData est chaîne)

:bUseMemo = Vrai

SI :nMemoCount > 0 ALORS
    // Recherche d'un "trou" pouvant etre utilisé
    M est entier
    POUR M=1 _A_ :nMemoCount
        SI :Memo[M]:nLin = 0 ALORS
            :Memo[M]:nLin = L
            :Memo[M]:nCol = C
            :Memo[M]:Data = sData
            RETOUR
        FIN
    FIN
FIN

:nMemoCount++

Dimension(:Memo,:nMemoCount)
:Memo[:nMemoCount]:nLin = L
:Memo[:nMemoCount]:nCol = C
:Memo[:nMemoCount]:Data = sData
```

Méthode MemoDel

```
PROCEDURE PUBLIQUE MemoDel(L est entier,C est entier=0,bLineDeleted est booléen=Vrai)

// Attention plusieurs mémo possibles donc parcours total

M est entier
POUR M=1 _A_ :nMemoCount
    SI :Memo[M]:nLin = L ALORS
        SI :Memo[M]:nCol = C OU C=0 ALORS
            :Memo[M]:nLin = 0
            :Memo[M]:nCol = 0
            :Memo[M]:Data = ""
        FIN
    FIN
    //Renumerotation des numéros de ligne associées si la ligne a été supprimée
    SI bLineDeleted ET :Memo[M]:nLin > L ALORS
        :Memo[M]:nLin = :Memo[M]:nLin-1
    FIN
FIN
```

Méthode MemoCompacte

```
PROCEDURE PUBLIQUE MemoCompacte()

// Premier à supprimer

nFirstMemoDel, M est entier
POUR M=1 _A_ :nMemoCount
    SI :Memo[M]:nLin = 0 ALORS
        nFirstMemoDel=M
        SORTIR
    FIN
    // Rien a compacter
    SI M=:nMemoCount ALORS
        RETOUR
    FIN
FIN

// Compactage (Retrait des lignes vides)
bDel est booléen
nbDeleted est entier

M=nFirstMemoDel
BOUCLE
    SI M > :nMemoCount OU M <= 0 ALORS
        SORTIR
    FIN

    SI :Memo[M]:nLin = 0 ALORS
        nbDeleted++
        bDel=Vrai
    SINON
        bDel=Faux
    FIN

    SI (PAS bDel) ET nbDeleted>0 ALORS
        :Memo[M-nbDeleted]:nLin = :Memo[M]:nLin
        :Memo[M-nbDeleted]:nCol = :Memo[M]:nCol
        :Memo[M-nbDeleted]:Data = :Memo[M]:Data
    FIN

    M++
FIN

// Suppression des memo inutiles en fin de tableau
POUR M=(:nMemoCount-nbDeleted+1) _A_ :nMemoCount
    :Memo[M]:nLin = 0
    :Memo[M]:nCol = 0
    :Memo[M]:Data = ""
FIN

:nMemoCount-=nbDeleted

Dimension(:Memo,:nMemoCount)

:bUseMemo = (:nMemoCount > 0)
```

Méthode MemoMove

```
PROCEDURE PUBLIQUE MemoMove(L est entier,C est entier,nL est entier,nC est entier=0)

M est entier
POUR M=1 _A_ :nMemoCount
    SI :Memo[M]:nLin = L ALORS
        SI C=0 ALORS
            :Memo[M]:nLin = nL
        SINON
            SI :Memo[M]:nCol = C  ALORS
                :Memo[M]:nLin = nL
                :Memo[M]:nCol = nC
            FIN
        FIN
    FIN
FIN
```

Méthode MemoSet

```
PROCEDURE PUBLIQUE MemoSet(L est entier,C est entier,Data est chaîne)

// Modification Memo si deja existant
M est entier
POUR M=1 _A_ :nMemoCount
    SI :Memo[M]:nLin = L ALORS
        SI :Memo[M]:nCol = C ALORS
            :Memo[M]:Data = Data
            RETOUR
        FIN
    FIN
FIN

:MemoAdd(L,C,Data)
```

Méthode InsertCol

```
FONCTION PUBLIQUE InsertCol(_AvantCol est entier,_Chaine est chaîne="")

SI _AvantCol > :nbLines ALORS
    RENVOYER :Add(_Chaine)
FIN

SI _AvantCol < 0 ALORS
    RENVOYER Faux
FIN

:JaugeMinMax(0,:nbLines)

L,C est entier

:nbCols++
Dimension(:Cell,:nbLines,:nbCols)

POUR L=1 _A_ :nbLines
    POUR C=:nbCols _A_ _AvantCol+1 PAS -1
        :Cell[L,C] = :Cell[L,C-1]

        SI :bUseMemo ALORS
            :MemoMove(L,C-1,L,C)
        FIN
    FIN

    :Cell[L,_AvantCol] = _Chaine

    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
FIN

:JaugeSet(0)

RENVOYER Vrai
```

Méthode MemoGet

```
FONCTION PUBLIQUE MemoGet(L est entier,C est entier=0)

sRes est chaîne

M est entier
POUR M=1 _A_ :nMemoCount
    SI :Memo[M]:nLin = L ALORS
        SI :Memo[M]:nCol = C ALORS
            sRes+=:Memo[M]:Data+RC
        FIN
    FIN
FIN

SI sRes <> "" ALORS
    sRes = Gauche(sRes,Taille(sRes)-2)
FIN

RENVOYER sRes
```

Méthode TrimCol

```
PROCEDURE TrimCol(_NumCol est entier,sCaract est chaîne=" ",bBefore est booléen=Vrai,bAfter est booléen=
Vrai)

L,N,C est entier
B est booléen

POUR L=1 _A_ :nbLines
    SI sCaract=" " ET bBefore ET bAfter ALORS
        :Cell[L,_NumCol]=SansEspace(:Cell[L,_NumCol])
    SINON
        SI bBefore ALORS
            C = Taille(sCaract)
            TANTQUE :Cell[L,_NumCol][[1 A C]] = sCaract
                :Cell[L,_NumCol] = :Cell[L,_NumCol][[C+1 A]]
            FIN
        FIN
        SI bAfter ALORS
            C = Taille(sCaract)
            N = Taille(:Cell[L,_NumCol])
            B = Faux
            TANTQUE :Cell[L,_NumCol][[N-C+1 sur C]] = sCaract
                B=Vrai
                N-=C
            FIN
            SI B ALORS
                :Cell[L,_NumCol] = :Cell[L,_NumCol][[1 A N]]
            FIN
        FIN
    FIN
FIN
```

Méthode TxtVersArbre

```
PROCEDURE TxtVersArbre(_NomArbre)

L est entier

POUR L=1 _A_ :nbLines
    ArbreAjoute(_NomArbre,:Line(L))
FIN
```

Méthode SwapLine

```
// Résumé : Echange les données de 2 lignes (Similaire à Move, meilleure méthode à vérifier)
// Syntaxe :
// SwapLine (<L> est entier, <L2> est entier)
//
// Paramètres :
//  L (entier) : <indiquez ici le rôle de L>
//  L2 (entier) : <indiquez ici le rôle de L2>
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE SwapLine(L est entier,L2 est entier)

SI L=L2 ALORS RETOUR

//A FAIRE:SwapLine Similaire à Move, meilleure méthode à vérifier

tLineTmp est un tableau dynamique = allouer un tableau de :nbCols chaînes

C est entier

POUR C=:nbCols A 1 PAS -1
    tLineTmp[C] = :Cell[L,C]
    :Cell[L,C] = :Cell[L2,C]
    :Cell[L2,C] = tLineTmp[C]
FIN
```

```
libérer tLineTmp

SI :bUseMemo ALORS
    :MemoMove(L,0,:nbLines+1)
    :MemoMove(L2,0,L)
    :MemoMove(:nbLines+1,0,L)
FIN
```

Méthode SortQuick

```
// Résumé : Procédure de Tri personnalisable
// Syntaxe :
// SortQuick ( [<_ByCol> est entier [, <_bSensPositif> est booléen [, <_bTriNumerique> est booléen [,
<_sFormatNum> est chaîne]]]])

//
// Paramètres :
//  _ByCol (entier - valeur par défaut=1) : <indiquez ici le rôle de _ByCol>
//  _bSensPositif (booléen - valeur par défaut=1) : <indiquez ici le rôle de _bSensPositif>
//  _bTriNumerique (booléen - valeur par défaut=0) : <indiquez ici le rôle de _bTriNumerique>
//  _sFormatNum (chaîne - valeur par défaut="012d") : <indiquez ici le rôle de _sFormatNum>
// Valeur de retour :
//  Aucune
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE SortQuick(_ByCol est entier=1,_bSensPositif est booléen=Vrai,_bTriNumerique est
booléen=Faux,_sFormatNum est chaîne="012d")

:SortQuick_Haere(1, :nbLines, _ByCol,_bSensPositif,_bTriNumerique,_sFormatNum)

SI PAS _bSensPositif ALORS
    :ReverseLines()
FIN
```

Méthode SortQuick_Haere

```
PROCEDURE PRIVÉE SortQuick_Haere(lo0, hi0, _ByCol=1,_bSensPositif=Vrai,_bTriNumerique=Faux,_sFormatNum=
"012d")

// Procédure basée sur :
// C.A.R/Tony Hoare's Quick Sort algorithm - http://www.wikipedia.org/wiki/C._A._R._Hoare

lo est un entier sans signe = lo0
hi est un entier sans signe = hi0

sRef est chaîne

SI PAS _bTriNumerique ALORS

    // Arbitrarily establishing partition element as the midpoint of the array.
    sRef = :Cell[PartieEntière((lo0+hi0) / 2),_ByCol]

    // loop through the array until indices cross
    TANTQUE( lo <= hi )

        // find the first element that is greater than or equal to
        // the partition element starting from the left Index.

        TANTQUE ( lo < hi0 ) ET ( :Cell[lo,_ByCol] < sRef )
            lo++
        FIN

        // find an element that is smaller than or equal to
        // the partition element starting from the right Index.
        TANTQUE ( hi > lo0 ) ET ( :Cell[hi,_ByCol] > sRef )
            hi--
        FIN

        // if the indexes have not crossed, swap
        SI lo <= hi ALORS
            :SwapLine(lo,hi)
```

```
            lo++
            hi--
        FIN

    FIN

SINON

    // Tri numérique
    sRef = NumériqueVersChaîne(:Cell[PartieEntière((lo0+hi0) / 2),_ByCol],_sFormatNum)

    TANTQUE( lo <= hi )
        TANTQUE ( lo < hi0 ) ET ( NumériqueVersChaîne(:Cell[lo,_ByCol],_sFormatNum) < sRef )
            lo++
        FIN
        TANTQUE ( hi > lo0 ) ET ( NumériqueVersChaîne(:Cell[hi,_ByCol],_sFormatNum) > sRef )
            hi--
        FIN
        SI lo <= hi ALORS
            :SwapLine(lo,hi)
            lo++
            hi--
        FIN
    FIN

FIN

// If the right index has not reached the left side of array must now sort the left partition.
SI lo0 < hi ALORS
    :SortQuick_Haere(lo0, hi, _ByCol,_bSensPositif,_bTriNumerique,_sFormatNum)
FIN

// If the left index has not reached the right side of array must now sort the right partition.
SI lo < hi0 ALORS
    :SortQuick_Haere(lo, hi0, _ByCol,_bSensPositif,_bTriNumerique,_sFormatNum)
FIN
```

Méthode Sort

```
// Résumé : Trie le tableau par les données de la colonne spécifiée
// Syntaxe :
//[ <Résultat> = ] Sort ( [<_ByCol> [, <_bSensPositif> est booléen [, <_bTriNumerique> est booléen [,
<_sFormatNum> est chaîne]]]])

//
// Paramètres :
//  _ByCol (valeur par défaut=1) : < indiquez ici le rôle de _ByCol >
//  _bSensPositif (booléen - valeur par défaut=1) : < indiquez ici le rôle de _bSensPositif >
//  _bTriNumerique (booléen - valeur par défaut=0) : < indiquez ici le rôle de _bTriNumérique >
//  _sFormatNum (chaîne - valeur par défaut="012d") : < indiquez ici le rôle de _sFormatNum >
// Valeur de retour :
//  booléen :   <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE Sort(_ByCol=1,_bSensPositif est booléen=Vrai,_bTriNumerique est booléen=Faux,_sFormatNum est
chaîne="012d")

SI _ByCol>:nbCols ALORS
    RENVOYER Faux
FIN

SI _bTriNumerique ALORS
    RENVOYER :SortOld(_ByCol,_bSensPositif,_bTriNumerique,_sFormatNum)
SINON
    RENVOYER :SortWD(_ByCol,_bSensPositif)
FIN

////Autre tri a revoir....
//// Algorithme de déplacement suivant index (peut etre deja breveté... si logique ;)
//
//:JaugeMinMax(0,:nbLines*2)
//
```

```
//L,C sont entiers
//
//nSrc,nDst,nCnt,nOrig sont entiers
//
////oNdx est un cTxt
////
////// Creation de l'index de tri
////POUR L=1 a :nbLines
////    Si _bTriNumérique ALORS
////        oNdx:Add(NumériqueVersChaine(Val(:Cell[L,_ByCol]),_sFormatNum)+TAB+L)
////    SINON
////        oNdx:Add(:Cell[L,_ByCol]+TAB+L)
////    FIN
////    SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
////FIN
////
////// Tri de l'index
////oNdx:Sort(1,_bSensPositif)
//
//// Création de la zone mémoire temporaire unique pour le tri
////-----------------------------------------------------------
//z est entier; zMem est chaine
//BOUCLE
//  zMem = "zcTab"+NumériqueVersChaine(z,"06d")
//  SI PAS MemExiste(zMem) ALORS SORTIR
//  z++
//FIN
//MemCrée(zMem)
////-----------------------------------------------------------
//
//POUR L=1 a :nbLines
//  Si _bTriNumérique ALORS
//      MemAjoute(zMem,"N"+NumériqueVersChaine(Val(:Cell[L,_ByCol]),_sFormatNum),L)
//  SINON
//      MemAjoute(zMem,:Cell[L,_ByCol],L)
//  FIN
//  SI (L modulo :nJaugeMinStep = 0) :JaugeSet(L)
//FIN
//MemTrie(zMem,_bSensPositif)
//
//tLineTmp est un tableau dynamique = Allouer un tableau de :nbCols chaines
//
////POUR nOrig=1 a oNdx:nbLines
////    SI Val(oNdx:Cell[nOrig,2]) <> nOrig ALORS
////        SORTIR
////    FIN
////FIN
//
//POUR nOrig=1 a :nbLines
//  SI Val(MemRecupère(zMem,nOrig)) <> nOrig ALORS
//      SORTIR
//  FIN
//FIN
//
//SI nOrig>:nbLines ALORS
//  //Tri non nécéssaire
//  RENVOYER Vrai
//FIN
//
//nSrc=nOrig
//
//BOUCLE
//
//  //Tantqu'on est pas revenu au point de départ
//
//  nCnt++
//
////    nDst = Val(oNdx:Cell[nSrc,2])
//  nDst = Val(MemRecupère(zMem,nSrc))
//  SI nDst=nOrig ALORS SORTIR
//
//  POUR C=1 a :nbCols
//      //Deplacement élement à destination
//      tLineTmp[C] = :Cell[nDst,C]
```

```
//      :Cell[nDst,C] = :Cell[nSrc,C]
//      :Cell[nSrc,C] = tLineTmp[C]
//  FIN
//
//  SI :bUseMemo ALORS
//      //Deplacement mémo à destination
//      :MemoMove(nSrc,0,:nbLines+1)
//      :MemoMove(nDst,0,nSrc)
//      :MemoMove(:nbLines+1,0,nSrc)
//  FIN
//
//  //:sSeparatorY = " "
//  //Trace(">"+:GetTxt())
//
//  nSrc = nDst
//
//  SI (nCnt modulo :nJaugeMinStep = 0) :JaugeSet(:nbLines+nCnt)
//
//FIN
//
//MemSupprimeTout(zMem)
//
//liberer tLineTmp
//
//:JaugeSet(0)
//
//RENVOYER Vrai
```

Méthode IniLit

```
// Résumé : Methode de substitution de IniLit sur les données du tableau
// Syntaxe :
//[ <Résultat> = ] IniLit (<_sSection> est chaîne [, <_sKey> est chaîne [, <_sValDef> est chaîne [,
<_sFicNotUsed> est chaîne]]])

//
// Paramètres :
//  _sSection (chaîne) : <indiquez ici le rôle de _sSection>
//  _sKey (chaîne) : <indiquez ici le rôle de _sKey>
//  _sValDef (chaîne) : <indiquez ici le rôle de _sValDef>
//  _sFicNotUsed (chaîne) : <indiquez ici le rôle de _sFicNotUsed>
// Valeur de retour :
//  Type indéterminé :              <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Attention, les sections ajoutées manuellement doivent être en majuscule !
//
FONCTION IniLit(LOCAL _sSection est chaîne,_sKey est chaîne="",_sValDef est chaîne="",_sFicNotUsed est
chaîne="")

_sSection=Majuscule(_sSection)
nTotLines, L est entier=TableauCherche(:Cell,tcLinéaire,1,_sSection)
nTotLines=:nbLines

//pour desactivation warning variable non utilisée
SI _sFicNotUsed="" ALORS
FIN

SI _sKey<>"" ALORS

    TANTQUE L>0 _ET_ L <=nTotLines

        SI :Cell[L,1] ~= _sSection ALORS
            SI :Cell[L,2] ~= _sKey ALORS
                RENVOYER :Cell[L,3]
            FIN
            L++
        SINON
            L=TableauCherche(:Cell,tcLinéaire,1,_sSection,L+1)
        FIN

    FIN

    RENVOYER _sValDef
```

```
SINON

    sRes est chaîne
    TANTQUE L>0 _ET_ L <= :nbLines

        SI :Cell[L,1] ~= _sSection ALORS
            sRes += :Cell[L,2] + RC
            L++
        SINON
            L=TableauCherche(:Cell,tcLinéaire,1,_sSection,L+1)
        FIN

    FIN

    RENVOYER sRes

FIN
```

Méthode IniEcrit

```
// Résumé : Methode de substitution de IniEcrit sur les données du tableau
// Syntaxe :
//[ <Résultat> = ] IniEcrit (<_sSection> est chaîne, <_sKey> est chaîne, <_sValue> est chaîne [,
<_sFicNotUsed> est chaîne])

//
// Paramètres :
//  _sSection (chaîne) : <indiquez ici le rôle de _sSection>
//  _sKey (chaîne) : <indiquez ici le rôle de _sKey>
//  _sValue (chaîne) : <indiquez ici le rôle de _sValue>
//  _sFicNotUsed (chaîne) : <indiquez ici le rôle de _sFicNotUsed>
// Valeur de retour :
//  Type indéterminé :                <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION IniEcrit(LOCAL _sSection est chaîne,_sKey est chaîne,LOCAL _sValue est chaîne,_sFicNotUsed est
chaîne="")

L est entier
nTotLig est entier=:nbLines

_sSection=Majuscule(_sSection)

L=TableauCherche(:Cell,tcLinéaire,1,_sSection)
SI L=-1 ALORS
    RENVOYER :AddLine(_sSection,_sKey,_sValue)
    //RENVOYER :Add(:FormatLine(_sSection,_sKey,_sValue))
FIN

POUR L=L _A_ nTotLig

    SI :Cell[L,1] ~= _sSection ALORS
        SI :Cell[L,2] ~= _sKey ALORS
            :Cell[L,3] = _sValue
            RENVOYER Vrai
        FIN
    FIN

FIN

//pour desactivation warning variable non utilisée
SI _sFicNotUsed="" ALORS
FIN

RENVOYER :Add(:FormatLine(_sSection,_sKey,_sValue))
```

Méthode ColOf

```
// Résumé : Renvoie la valeur d'une colonne (pratique dans les parcours :Premier()/:Suivant())
// Syntaxe :
// [ <Résultat> = ] ColOf (<C> est entier [, <L> est entier])
//
// Paramètres :
//  C (entier) : <indiquez ici le rôle de C>
//  L (entier - valeur par défaut=:EnCours) : <indiquez ici le rôle de L>
// Valeur de retour :
//  Type indéterminé : <indiquez ici les valeurs possibles ainsi que leur interprétation>
//
// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE ColOf(C est entier,L est entier=:EnCours)

QUAND EXCEPTION
    ExceptionActive()
    RENVOYER EOT
FIN

RENVOYER Remplace(:Cell[L,C],EOT,"")
```

Méthode Positionne

```
// Résumé : Positionne le curseur :EnCours
// Syntaxe :
//Positionne (<_L> est entier)
//
// Paramètres :
//  _L (entier) : <indiquez ici le rôle de _L>
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE Positionne(_L est entier)

SI _L<=:nbLines ALORS
    :EnCours = _L
    :EnDehors = Faux
SINON
    :EnDehors = Vrai
FIN
```

Méthode FindInLine

```
// Résumé : Recherche du texte dans une SEULE ligne ou une plage de colonnes de la ligne spécifiée
// Syntaxe :
//[ <Résultat> = ] FindInLine (<_sRech> est chaîne [, <_nLine> est entier [, <_Options> est entier [,
<_nColMin> est entier [, <_nColMax> est entier]]]])

//
// Paramètres :
//  _sRech (chaîne) : <indiquez ici le rôle de _sRech>
//  _nLine (entier) : Indice de la ligne dans laquelle recherche
//  _Options (entier - valeur par défaut=0) : < indiquez ici le rôle de _Options >
//  _nColMin (entier - valeur par défaut=1) : < indiquez ici le rôle de _nColMin >
//  _nColMax (entier - valeur par défaut=:nbCols) : < indiquez ici le rôle de _nColMax >
// Valeur de retour :
//  entier :   Le Résultat est le numéro de ligne
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE FindInLine(_sRech est chaîne,_nLine est entier=:EnCours,_Options est entier=0,_nColMin
 est entier=1,LOCAL _nColMax est entier=:nbCols)

L,C sont entiers

L = _nLine

SI _nColMax=0 ALORS
```

```
    _nColMax=:nbCols
FIN

POUR C=_nColMin A _nColMax
    SELON _Options
        CAS 0
            //Recherche Stricte
            SI :Cell[L,C] = _sRech ALORS
                RENVOYER C
            FIN
        CAS SansCasse
            //Recherche sans attention aux Min/Maj et Espaces
            SI :Cell[L,C] ~= _sRech ALORS
                RENVOYER C
            FIN
        AUTRES CAS
            //MotComplet/DepuisDebut/DepuisFin (ou combinaisons)
            SI Position(:Cell[L,C],_sRech,1,_Options) > 0 ALORS
                RENVOYER C
            FIN
    FIN
FIN

RENVOYER 0
```

Méthode FindInHead

```
FONCTION PUBLIQUE FindInHead(_sRech est chaîne,_Options est entier=0,_nColMin est entier=1,LOCAL
_nColMax est entier=:nbCols)

// Le Résultat est le numéro de ligne
C sont entiers

SI _nColMax=0 ALORS
    _nColMax=:nbCols
FIN

POUR C=_nColMin A _nColMax
    SELON _Options
        CAS 0
            //Recherche Stricte
            SI :Head[C] = _sRech ALORS
                RENVOYER C
            FIN
        CAS SansCasse
            //Recherche sans attention aux Min/Maj et Espaces
            SI :Head[C] ~= _sRech ALORS
                RENVOYER C
            FIN
        AUTRES CAS
            //MotComplet/DepuisDebut/DepuisFin (ou combinaisons)
            SI Position(:Head[C],_sRech,1,_Options) > 0 ALORS
                RENVOYER C
            FIN
    FIN
FIN

RENVOYER 0
```

Méthode ParseStream

```
PROCEDURE PUBLIQUE ParseStream(_Data)

bBefore est un booléen = :bCropSource
:bCropSource = Vrai

:Parse(_Data)

:bCropSource = bBefore
```

Méthode DropDoubles

```
// Résumé : Retire les doublons sur la colonne spécifiée ou une plage de colonne, conserve le premier
// Syntaxe :
//[ <Résultat> = ] DropDoubles ( [<nCol> est entier [, <bFast> est booléen [, <nLastCol> est entier]]])
//
// Paramètres :
//  nCol (entier - valeur par défaut=1) : Colonne des clés
//
bFast (booléen - valeur par défaut=0) : A Vrai (plus rapide) uniquement si le contenu est déjà trié par
la colonne de dédoublonnage

//
nLastCol (entier - valeur par défaut=0) : Attention, par défaut dédoublonnage sur une seule colonne, mais
modifiable ici pour une plage de colonnes

// Valeur de retour :
//  booléen :      //   Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PUBLIQUE DropDoubles(nCol est entier=1,bFast est booléen=Faux,nLastCol est entier=0)

SI nCol>:nbCols ALORS
    RENVOYER Faux
FIN

SI nLastCol=0 ALORS
    nLastCol=nCol
FIN

L,L2,LL,nbDbl est entier
S1,S2 sont chaînes
:DeleteOptBegin()

SI bFast ALORS

    L=:nbLines
    nbDbl=1
    LL=L-nbDbl
    //On considère que le tri est deja effectué sur la colonne
    TANTQUE L > 1 ET LL >= 1

        S1=:Cell[LL,nCol]
        S2=:Cell[L,nCol]
        SI nLastCol<>nCol ALORS
            S1=:Line(LL,nCol,nLastCol)
            S2=:Line(L,nCol,nLastCol)
        FIN

        SI S1=S2 ALORS
            :Del(L)
            :nFirstLineDel = L
            nbDbl++
        SINON
            nbDbl=1
            L=LL
        FIN

        //précédent
        LL=L-nbDbl

    FIN

SINON

    POUR L=:nbLines A 1 PAS -1
        SI nLastCol=nCol ALORS
            S1=:Cell[L,nCol]
        SINON
            S1=:Line(L,nCol,nLastCol)
        FIN
        POUR L2=1 A :nbLines
            SI nLastCol=nCol ALORS
                S2=:Cell[L2,nCol]
```

```
            SINON
                S2=:Line(L2,nCol,nLastCol)
            FIN
            SI L<>L2 ET S1=S2 ALORS
                :Del(L)
                :nFirstLineDel = L
            FIN
        FIN
    FIN

FIN
:DeleteOptEnd()

RENVOYER Vrai
```

Méthode GetCols

```
FONCTION PUBLIQUE GetCols(_nColDeb est entier=1,_nColFin est entier=:nbCols,bWithEntete est booléen=Faux
)

// Renvoie le tableau sous forme de chaine

sRes est chaîne
L, nTotLig sont entier

SI bWithEntete ALORS
    Sres += :GetEntete(_nColDeb,_nColFin)+:sSeparatorY
FIN

nTotLig=:nbLines
POUR L=1 _A_ nTotLig
    Sres += :Line(L,_nColDeb,_nColFin)+:sSeparatorY
FIN

SI Sres <> "" ALORS
    Sres = Gauche(Sres,Taille(Sres)-Taille(:sSeparatorY))
FIN


RENVOYER sRes
```

Méthode TxtVersFormatSpec

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//[ <Résultat> = ] TxtVersFormatSpec (<_sFormatExport> est chaîne [, <_nVerifTaille> est entier [,
<_nRenvoiNbLignes>]])

//
// Paramètres :
//  _sFormatExport (chaîne) : <indiquez ici le rôle de _sFormatExport>
//
_nVerifTaille (entier - valeur par défaut=0) : Taille complète de la ligne, pour vérification rapide
(actif si <> 0)

//
_nRenvoiNbLignes (valeur par défaut=-1) : Variable passée par adresse pour obtenir le nombre de lignes
exportées

// Valeur de retour :
//  chaîne :  <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
//  sFormatExport est chaine = ...
//  "N"+TAB+"5d"+RC+...    // N  de Mouvement
//  "A"+TAB+"2"+RC+...     // Journal
//  "D"+TAB+"8"+RC+...     // Date d'écriture
//  "D"+TAB+"8"+RC+...     // Date d'échéance
//  "A"+TAB+"12"+RC+...    // N  de Piece
//  "A"+TAB+"11"+RC+...    // N  de Compte
//  "A"+TAB+"25"+RC+...    // Libellé de l'écriture
//  "N"+TAB+"13.2f"+RC+... // Montant
//  "A"+TAB+"1"+RC+...     // Sens de l'opération Débit ou Crédit D/C
```

```
//  "A"+TAB+"12"+RC+...    // N  de Pointage
//  "A"+TAB+"6"+RC+...     // Code analytique/budgetaire
//  "A"+TAB+"34"+RC+...    // Libellé du compte (facultatif)
//  "A"+TAB+"1"+RC+...     // Euro O/N
//  "A"+TAB+"4"            // Version 2003

PROCEDURE TxtVersFormatSpec(_sFormatExport est chaîne,_nVerifTaille est entier=0,_nRenvoiNbLignes=-1)


oFormat est un cTxt(_sFormatExport)
sData, S, sRes est chaîne
L, C, N, nTotLig, nTotCol sont des entiers
dtDate est une Date
dtTime est une Heure
nTotLig = :nbLines
nTotCol = :nbCols

POUR L=1 _A_ nTotLig
    S=""
    POUR C=1 A nTotCol
        sData = :Cell[L,C]
        SELON Majuscule(oFormat:Cell[C,1])
            CAS "A"
                N=Val(oFormat:Cell[C,2])
                S+=Complète(Gauche(sData,N),N)
            CAS "D"
                dtDate = sData
                N=Val(oFormat:Cell[C,2])
                S+=Complète(Gauche(dtDate,N),N)
            CAS "T"
                dtTime = sData
                N=Val(oFormat:Cell[C,2])
                S+=Complète(Gauche(dtTime,N),N)
            CAS "N"
                S+=NumériqueVersChaîne(Val(sData),oFormat:Cell[C,2])
        FIN
    FIN
    SI _nVerifTaille<>0 ALORS
        SI _nVerifTaille<>Taille(S) ALORS
            RENVOYER "ERREUR SUR VERIFICATION TAILLE DE LIGNE"
        FIN
    FIN
    sRes+=S+RC
FIN

SI sRes<>"" ALORS
    sRes=Gauche(sRes,Taille(sRes)-2)
FIN

SI _nRenvoiNbLignes<>-1 ALORS
    _nRenvoiNbLignes=:nbLines
FIN

RENVOYER sRes
```

Méthode SortWD

```
PROCEDURE SortWD(_ByCol=1,_bSensPositif est booléen=Vrai,*)
//_bTriNumerique est booléen=Faux,_sFormatNum est chaîne="012d"

SI _ByCol>:nbCols ALORS
    RENVOYER Faux
FIN

L,C sont entiers

SI :bUseMemo ALORS
    //On ajoute un colonne pour conserver les anciens indices
    C=:AddCol()
    POUR L=1 _A_ :nbLines
        :Cell[L,C]=L
    FIN
FIN
```

```
SI _ByCol=1 ALORS
    SI _bSensPositif ALORS
        TableauTrie(:Cell,ttCroissant)
    SINON
        TableauTrie(:Cell,ttDécroissant)
    FIN
SINON
    //Le tri peut etre sur des colonnes multiples (voir doc TableauTrie)
    sTri est chaîne=_ByCol
    TableauTrie(:Cell,ttColonne,sTri)
FIN

SI :bUseMemo ALORS
    POUR L=1 _A_ :nbLines
        :MemoMove(L,0,Val(:Cell[L,C]))
    FIN
    //"retrait" derniere colonne
    :nbCols--
FIN

RENVOYER Vrai
```

Méthode ExtractCSV

```
// Résumé : Convertit les séparateurs entre guillemets de cellule CSV en marqueurs spécifiques internes
(¤TAB¤,¤RC¤)

// Syntaxe :
//ExtractCSV (<S> est chaîne)
//
// Paramètres :
//  S (chaîne) : <indiquez ici le rôle de _Data>
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE PROTÉGÉE ExtractCSV(S est chaîne)

sDQ est chaîne = Caract(34) // (")

S=Remplace(S,sDQ+:sSeparatorX+sDQ,"¤X¤")

S=Remplace(S,sDQ+:sSeparatorY+sDQ,"¤Y¤")
//_Data=Remplace(_Data,:sSeparatorY+sDQ,"¤Y¤")
//_Data=Remplace(_Data,sDQ+:sSeparatorY,"¤Y¤")

S=Remplace(S,:sSeparatorX,"¤TAB¤")
S=Remplace(S,:sSeparatorY,"¤RC¤")

S=Remplace(S,"¤Y¤",:sSeparatorY)
S=Remplace(S,"¤X¤",:sSeparatorX)

//En mode CSV avec Commentaire le dernier séparateur Y pose probleme (début de fichier géré uniquement
pour les commentaires)

nD, nRC sont entier

SI :sIgnoredLinesComment<>"" ALORS

    nD=Position(S,"¤RC¤"+:sIgnoredLinesComment,Taille(S),DepuisFin)
    nRC=Position(S,"¤RC¤",nD+4+Taille(:sIgnoredLinesComment))
    SI nD>0 ET nRC>0 ALORS
          //si ¤RC¤ avant vrai RC (ou sepY)
            SI nRC < Position(S,:sSeparatorY,nD+4+Taille(:sIgnoredLinesComment)) ALORS
                S=S[[A nRC-1]]+:sSeparatorY+S[[nRC+4 A]]
            FIN
    FIN

    S=Remplace(S,"¤RC¤"+:sIgnoredLinesComment,:sSeparatorY+:sIgnoredLinesComment)

FIN

nRC=Position(S,"¤RC¤""")
```

```
//Gestion du premier guillemet
SI nRC ET PAS Position(S,"¤RC¤",nRC-5,DepuisFin) ALORS
    S=S[[A nRC-1]]+:sSeparatorY+S[[nRC+5 A]]
FIN

//Gestion du dernier guillemet
SI Droite(S,1)=Caract(34) ALORS
    S=S[[A Taille(S)-1]]
SINON
    nRC=Position(S,"""¤RC¤",Taille(S),DepuisFin)
    SI nRC ET PAS Position(S,"¤RC¤",nRC+5) ALORS
        S=S[[A nRC-1]]+:sSeparatorY+S[[nRC+5 A]]
    FIN
FIN
```

Méthode TxtVersExcel

```
// Résumé : Exporte vers Excel
// Syntaxe :
//TxtVersExcel (<sFileName> est chaîne [, <bQuitAfter> [, <bWithEntete> [, <nLineMin> [, <nLineMax> [,
<nColMin> [, <nColMax>]]]]]])

//
// Paramètres :
//  sFileName (chaîne) : <indiquez ici le rôle de sFileName>
//  bQuitAfter (valeur par défaut=1) : < indiquez ici le rôle de bQuitAfter >
//  bWithEntete (valeur par défaut=0) : < indiquez ici le rôle de bWithEntete >
//  nLineMin (valeur par défaut=1) : < indiquez ici le rôle de nLineMin >
//  nLineMax : <indiquez ici le rôle de nLineMax>
//  nColMin (valeur par défaut=1) : < indiquez ici le rôle de nColMin >
//  nColMax : <indiquez ici le rôle de nColMax>
// Valeur de retour :
//  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.

PROCEDURE TxtVersExcel(sFileName est chaîne,bQuitAfter=Vrai,bWithEntete=Faux,nLineMin=1,nLineMax=:
nbLines,nColMin=1,nColMax=:nbCols)
```

```
// Retrait warning, mais a implementer (procedure a revoir)
SI bWithEntete ALORS
FIN

// Création d'un  objet automation dynamique
EXCEL est un objet Automation dynamique

// Instance de Excel existante ?
SI ObjetActif("Excel.Application") <> Null ALORS
    // Récupération de l'instance de Excel existante
    EXCEL = ObjetActif("Excel.Application")
SINON
    // Allocation de l'objet automation "EXCEL"
    EXCEL = allouer un objet Automation "Excel.Application"
FIN

EXCEL>>Visible = Faux
// Ouverture d'un classeur d'Excel
EXCEL>>Workbooks>>Add()

nAsc est entier = Asc("A")-1
L, C sont des entiers
POUR L=nLineMin _A_ nLineMax
    POUR C=nColMin _A_ nColMax
        EXCEL>>Range(Caract(nAsc+C)+L)>>Value = :CellGet(L,C,Vrai)
    FIN
FIN

EXCEL>>ActiveWorkBook>>SaveAs(sFileName)

SI bQuitAfter ALORS
```

```
    EXCEL>>Quit()
SINON
    EXCEL>>Visible = Vrai
FIN
```

```
//Pour changer de feuille,:
//  EXCEL>>Sheets("Feuil2")>>Select()
```

Méthode CellGet

```
// Résumé : Récupère une cellule, sans erreur possible (comme :Cell[L,C]) par défaut la colonne 1 de la
ligne :EnCours

// Syntaxe :
//[ <Résultat> = ] CellGet ( [<L> est entier [, <C> est entier [, <bNoEOT> est booléen]]])
//
// Paramètres :
//  L (entier) : <indiquez ici le rôle de L>
//  C (entier - valeur par défaut=1) : <indiquez ici le rôle de C>
//  bNoEOT (booléen - valeur par défaut=0) : < indiquez ici le rôle de bNoEOT >
// Valeur de retour :
//  Type indéterminé :            <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
FONCTION PUBLIQUE CellGet(L est entier=:EnCours,C est entier=1,bNoEOT est booléen=Faux)

SI L=0 OU L>:nbLines OU C>:nbCols ALORS
    RENVOYER ""
FIN

SI bNoEOT ALORS
    RENVOYER Remplace(:Cell[L,C],EOT,"")
SINON
    RENVOYER :Cell[L,C]
FIN
```

Méthode IniVersTxt_Linux

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//[ <Résultat> = ] IniVersTxt_Linux (<_FichierIni> est chaîne [, <_Section> est chaîne [, <_bMerge> est
booléen]])

//
// Paramètres :
//  _FichierIni (chaîne) : <indiquez ici le rôle de _FichierIni>
//  _Section (chaîne) : <indiquez ici le rôle de _Section>
//  _bMerge (booléen - valeur par défaut=0) : <indiquez ici le rôle de _bMerge>
// Valeur de retour :
//  booléen :      //   Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE IniVersTxt_Linux(_FichierIni est chaîne,_Section est chaîne="",_bMerge est booléen=Faux)

sINI est chaîne = fChargeTexte(_FichierIni)

sINI=Remplace(sINI,Caract(10),RC)
sINI=Remplace(sINI,Caract(13),"")

sLine est chaîne
sSection est chaîne
POUR TOUTE CHAINE sLine DE sINI SEPAREE PAR Caract(10)
    SI PAS sLine~~"" _ET_ sLine[[1]]<>";" ALORS
        SI sLine[[1]]="[" ALORS
            sSection=sLine[[2 A Position(sLine,"]")-1]]
        SINON
```

```
            SI _Section="" _OU_ sSection~=_Section ALORS
                SI _bMerge ALORS
                    :IniEcrit(sSection,SansEspace(ExtraitChaîne(sLine,1,"=")),SansEspace(ExtraitChaîne(
                    sLine,2,"=")))
                SINON
                    :Add(sSection+:sSeparatorX+SansEspace(ExtraitChaîne(sLine,1,"="))+:sSeparatorX+
                    SansEspace(ExtraitChaîne(sLine,2,"=")))
                FIN
            FIN
        FIN
    FIN
FIN
RENVOYER Vrai
```

Méthode AddLine

```
// Résumé : TableauAjouteLigne
// Syntaxe :
//[ <Résultat> = ] AddLine ([...])
//
// Paramètres :
//  Aucun
// Valeur de retour :
//  entier :        //  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE AddLine(*)

//v3.32
SI :nbCols < MesParamètres..Occurrence ALORS
    :Resize(Max(:nbLines,1),MesParamètres..Occurrence)
FIN

SI TableauInfo(:Cell,tiNombreLignes) > :nbLines ALORS
    :Resize(Max(:nbLines,1),MesParamètres..Occurrence)
FIN

SI :nbLines=1 _ET_ :Line(1) = "" ALORS
    TableauSupprimeTout(:Cell)
    :nbLines=0
FIN

nRes est entier = TableauAjouteLigne(:Cell,MesParamètres)

POUR C=MesParamètres..Occurrence+1 _A_ :nbCols
    :Cell[nRes,C]=EOT
FIN

SI nRes>0 ALORS
    :nbLines++
FIN

RENVOYER nRes
```

Méthode TxtVersHtml

```
// Résumé : <indiquez ici ce que fait la procédure>
// Syntaxe :
//[ <Résultat> = ] TxtVersHtml ( [<bHeadToo> est booléen [, <tAttributs>]])
//
// Paramètres :
//  bHeadToo (booléen - valeur par défaut=0) : <indiquez ici le rôle de bHeadToo>
//  tAttributs (valeur par défaut=0) : <indiquez ici le rôle de sClass>
// Valeur de retour :
//  chaîne :                <indiquez ici les valeurs possibles ainsi que leur interprétation>
//// Exemple :
PROCEDURE PUBLIQUE TxtVersHtml(bHeadToo est booléen=Faux, tAttributs=Null)

sRes est chaîne

sTable est chaîne = "<table>"
sTR est chaîne = "<tr>"
```

```
sTD est chaîne = "<td>"
sTH est chaîne = "<th>"

sTR2 est chaîne = "<tr>"
sTD2 est chaîne = "<td>"

bTR2 est boolean
bTD2 est boolean

SI TypeVar(tAttributs) = wlTableauAssociatif ALORS
    sTable = "<table"+[" "]+tAttributs["table"]+">"
    sTR = "<tr"+[" "]+tAttributs["tr"]+">"
    sTD = "<td"+[" "]+tAttributs["td"]+">"
    sTH = "<th"+[" "]+tAttributs["th"]+">"
    SI tAttributs["tr2"]<>"" ALORS
        bTR2 = Vrai
        sTR2 = "<tr"+[" "]+tAttributs["tr2"]+">"
    FIN
    SI tAttributs["td2"]<>"" ALORS
        bTD2 = Vrai
        sTD2 = "<td"+[" "]+tAttributs["td2"]+">"
    FIN
FIN

sCell est chaîne
L,C sont entier
nTotLig est entier=:nbLines
nTotCol est entier=:nbCols

sRes = sTable

SI bHeadToo ALORS
    sRes+=sTR
    POUR C=1 A nTotCol
        sRes+=sTH+:Head[C]+"</th>"
    FIN
    sRes+="</tr>"+RC
FIN

POUR L=1 A nTotLig
    SI bTR2 _ET_ (L modulo 2)=0 ALORS
        //Lignes paires
        sRes+=sTR2
    SINON
        sRes+=sTR
    FIN

    SI bTD2 _ET_ (L modulo 2)=0 ALORS
        //Lignes paires
        POUR C=1 A nTotCol
            sCell = :CellGet(L,C,Vrai)
            SI sCell<>"" ALORS
                sRes+=sTD2+sCell+"</td>"
            SINON
                sRes+=sTD2+" </td>"
            FIN
        FIN
    SINON
        POUR C=1 A nTotCol
            sCell = :CellGet(L,C,Vrai)
            SI sCell<>"" ALORS
                sRes+=sTD+sCell+"</td>"
            SINON
                sRes+=sTD+" </td>"
            FIN
        FIN
    FIN

    sRes+="</tr>"+RC
FIN

RENVOYER sRes+"</table>"
```

Méthode AssocVersTxt

```
// Résumé : Import de Tableau Associatif
// Syntaxe :
//[ <Résultat> = ] AssocVersTxt (<tTab> [, <bAjout> est booléen])
//
// Paramètres :
//  tTab : <indiquez ici le rôle de tTab>
//  bAjout (booléen - valeur par défaut=0) : <indiquez ici le rôle de bAjout>
// Valeur de retour :
//  entier :    //  Aucune
//// Exemple :
// Indiquez ici un exemple d'utilisation.
//
PROCEDURE AssocVersTxt(tTab,bAjout est boolean=Faux)

sData,sKey est chaîne
L est entier

SI PAS bAjout ALORS
    :Resize(TableauOccurrence(tTab), 2)
SINON
    L=:nbLines
    :Resize(:nbLines+TableauOccurrence(tTab), Max(2,:nbCols))
FIN

POUR TOUT ELEMENT Data, Key DE tTab
    L++
    sData=Data
    sKey=Key
    :Cell[L,1]=sKey
    :Cell[L,2]=sData
FIN

RENVOYER :nbLines
```

Méthode cTxtVersTxt

```
// Résumé : Import d'objet cTxt
// Syntaxe :
//[ <Résultat> = ] cTxtVersTxt (<oTxt> est cTxt [, <bAjout> est booléen])
//
// Paramètres :
//  oTxt (cTxt) : <indiquez ici le rôle de tTab>
//  bAjout (booléen - valeur par défaut=0) : <indiquez ici le rôle de bAjout>

PROCEDURE cTxtVersTxt(oTxt est un cTxt,bAjout est boolean=Faux)

SI PAS bAjout ALORS
    :Resize(oTxt:nbLines, oTxt:nbCols)
    TableauCopie(oTxt:Cell,:Cell)
    :Resize(oTxt:nbLines, oTxt:nbCols)
SINON
    TableauAjoute(:Cell,oTxt:Cell)
    :Resize(:nbLines+oTxt:nbLines, :nbCols)
FIN

RENVOYER :nbLines
```