100% COURS
MCours.com
Cours et mémoires gratuits
mcourscom@gmail.com

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

I - Introduction

Les variables temporaires font parties des nombreuses nouveautés apparues avec la version 2007 de Microsoft Access. Si leur intérêt est parfois négligé par bon nombre de développeurs, je les trouve, pour ma part, particulièrement pratiques notamment lors d'échange de données entre différents applicatifs.

En détail, il s'agit en fait de variables disponibles dans n'importe quel module d'une application (un peu comme les variables déclarées en Public) et qui offrent l'avantage d'être aussi accessibles depuis un autre projet via automation par exemple. Bien entendu, comme toute porte vers l'extérieur cela représente une faille de sécurité et ne doit être réservé qu'à des échanges et des stockages non critiques.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

II - Manipulation

Les variables temporaires **TempVar** sont des objets regroupés au sein d'une collection **Application.TempVars** qui vous permet par conséquent de créer autant de variables temporaires que nécessaire sans toutefois dépasser la limite de 256. Attention cependant, il ne faut pas que l'utilisation des variables temporaires devienne une solution de facilité et systématique sans quoi vous risquez de vous retrouver avec des dizaines de variables inutiles transformant votre développement en usine à gaz.

La collection **TempVars** permet d'ajouter, de modifier et de supprimer des variables temporaires. A l'ouverture de l'application, aucune variable n'est disponible (Access n'en utilise pas par défaut, il s'agit d'une fonctionnalité réservée au développeur).

Chaque variable possède un nom et une valeur de type **Variant**. L'ajout est réalisé en invoquant la méthode **Add** de la collection.

Exemple :

```
Sub ajout()
    'Crée une nouvelle variable
    Application.TempVars.Add "Ma Variable", 100
    'Affiche la variable
    MsgBox Application.TempVars("Ma Variable").Value
End Sub
```

Ici, le nom de la variable est *Ma Variable* et sa valeur est 100. Comme vous pouvez le constater, l'accès à la variable se fait ensuite comme pour toute collection, c'est-à-dire en spécifiant directement le nom de l'objet initialement défini.

Il est aussi possible d'utiliser l'index de l'objet **TempVar** dans la collection.

Exemple :

```
MsgBox Application.TempVars(0).Value
```

Pour modifier le contenu d'une variable temporaire deux techniques peuvent être utilisées :

1. Invoquer la méthode **Add** de nouveau pour écraser l'ancienne valeur
2. Modifier la propriété Value de l'objet **TempVar** concerné

Premier exemple :

```
'Crée une nouvelle variable nommée Ma Variable
    Application.TempVars.Add "Ma Variable", 100
    'Ecrase la variable pour modifier son contenu
    Application.TempVars.Add "Ma Variable", 105
    'Affiche la variable
    MsgBox Application.TempVars("Ma Variable").Value
```

Deuxième exemple :

```
    'Crée une nouvelle variable nommée Ma Variable
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
    Application.TempVars.Add "Ma Variable", 100
    'Modifie la propriété Value
    Application.TempVars("Ma Variable").Value = 105
    'Affiche la variable
    MsgBox Application.TempVars("Ma Variable").Value
```

Je vous recommande d'utiliser toujours la deuxième méthode afin d'éviter d'écraser des variables par mégarde et de perdre ainsi du temps en *déboguage*.

La méthode **Remove** permet de supprimer une variable temporaire en spécifiant son nom (ou son index) en argument. La méthode **RemoveAll** quant à elle permet de supprimer l'ensemble des variables temporaires.

```
    'Crée une nouvelle variable nommée Ma Variable
    Application.TempVars.Add "Ma Variable", 100
    'Supprime la variable
    Application.TempVars.Remove "Ma Variable"
```

Attention, si vous tentez de supprimer une variable qui n'existe pas, aucune erreur ne sera générée. De la même façon, si vous essayez d'accéder à une variable inexistante la valeur retournée sera **Null**. Cette trop grande permissivité est à regretter et donne un peu l'impression d'une fonctionnalité mal finie.

Enfin, comme pour toute collection, la propriété **Count** permet de connaître le nombre de variables temporaires utilisées :

```
Dim i As Integer
For i = 1 To 10
    Application.TempVars.Add "MaVariable" & i, i
Next i
MsgBox Application.TempVars.Count
```

En résumé

Add	**Méthode**. Ajoute la variable temporaire dont le nom et la valeur sont passés en paramètre.
Count	**Propriété (Long)**. Retourne le nombre de variables temporaires créées.
Item	**Méthode**. Retourne l'objet **TempVar** correspondant dont le nom ou l'indice est passé en paramètre.
Remove	**Méthode**. Supprime la variable temporaire dont le nom ou l'indice est passé en paramètre.
RemoveAll	**Méthode**. Supprime toutes les variables temporaires.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

III - Utilisation des variables temporaires dans un formulaire

Il est possible d'afficher directement la valeur d'une variable temporaire dans un contrôle en plaçant la syntaxe d'appel dans la propriété **Source** du contrôle concerné.

Par exemple, dans le cadre d'une zone de texte affichant l'heure à laquelle l'application a été ouverte :

1. La macro **Autoexec** renseigne la variable temporaire **tmpDateHeure** avec le code VBA suivant :

```
Function AutoExec()
    Application.TempVars.Add "tmpDateHeure", Format(Now, "dd mmmm yyyy à hh:nn:ss")
End Function
```

2. Le formulaire possède une zone de texte dont la **source** est :

```
=[Application].[VarTemp]("tmpDateHeure")
```

*Le générateur d'expression traduit **Application.TempVars** en **[Application].[VarTemp]** automatiquement.*

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

IV - Utilisation des variables temporaires dans une requête

Si vous souhaitez utiliser la valeur d'une variable temporaire dans une requête, vous ne pouvez pas utiliser une syntaxe similaire à celle vue juste avant. Vous devez obligatoirement passer par une fonction VBA qui sera chargée de retourner le contenu de la variable passée en paramètre. Vous intégrerez ensuite cette fonction dans votre SQL.

```
Function RetourTempVar(strTempVar As String) As Variant
    RetourTempVar = Application.TempVars(strTempVar)
End Function
```

Le code SQL qui aurait pu être :

```
SELECT [Application].[VarTemp]("tmpDateHeure")
FROM MaTable
```

Devient :

```
SELECT RetourTempVar("tmpDateHeure")
FROM MaTable
```

Pour les habitués, il s'agit d'un procédé identique à celui employé pour utiliser la fonction **Replace** dans les requêtes d'Access 2000

100% COURS
MCours.com
Cours et mémoires gratuits
mcourscom@gmail.com

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

V - Mise en évidence dans le cadre d'automation

Pour illustrer l'échange de données entre deux applications à l'aide des variables temporaires nous allons créer deux fichiers Microsoft Access. Le premier se contentera d'afficher l'état d'une variable temporaire, le second sera chargé de modifier régulièrement les variables temporaires du précédent et de lancer l'affichage (l'**API** Windows **Sleep** permettra de temporiser).

V-A - Client.accdb

Créez une nouvelle base de données **client.accdb** et ajoutez-y un module nommé **mduClient** contenant le code suivant :

```
Option Compare Database


Function Afficher()
    With Application
        'Teste si la variable temporaire "Ma Variable" existe
        If Not IsNull(.TempVars("Ma Variable").Value) Then
            MsgBox .TempVars("Ma Variable").Value
        End If
    End With
End Function
```

V-B - Serveur.accdb

Nous allons procéder de la même façon avec un module nommé **mduServeur** contenant le code suivant :

```
Option Compare Database


Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Function Dialoguer()
'Déclarartion de la variable Application correspondant à Client.accdb
Dim oCliApp As Access.Application
'Crée une nouvelle instance d'Access
Set oCliApp = New Access.Application
'Ouvre le fichier client.accdb
oCliApp.OpenCurrentDatabase ("D:\client.accdb")

With oCliApp
    'Crée la variable temporaire
    .TempVars.Add "Ma Variable", 0

    'Toutes les 2 secondes, le programme va modifier la variable temporaire
    'Ma Variable dans client.accdb
    While True
        .TempVars("Ma Variable") = .TempVars("Ma Variable") + 1
        oCliApp.Run "Afficher"
        DoEvents
        Sleep 2000
    Wend
End With
End Function
```

Résultats :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VI - Programmation Orientée Objet

Jusque là, nous avons utilisé des variables numériques. La propriété **Value** d'un objet **TempVar** étant **Variant**, il est en fait possible de stocker n'importe quelle variable d'un sous-type de **Variant** : **Integer**, **Currency**, **String**, **Single**, **Boolean**, etc. (y compris la valeur **NULL**). En revanche si vous tentez d'y stocker un objet vous rencontrer l'erreur **32538** : *Les variables temporaires ne peuvent contenir que des données, pas des objets*.

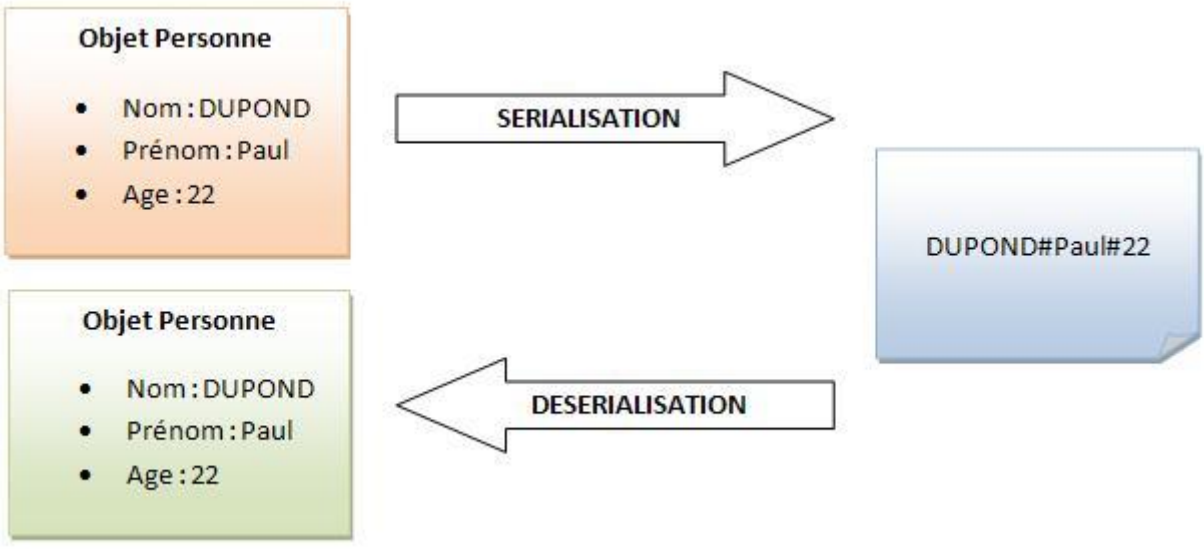
Alors comment procéder pour stocker un objet ? Bien souvent ce qui est intéressant de conserver ou d'échanger dans un objet ne sont pas ses méthodes (facilement réutilisable depuis un autre objet de la même classe) mais plutôt la valeur de ses propriétés. Pour cela deux cas de figures :

- Stocker chaque propriété dans un objet **TempVar** correspondant. Cette solution peut s'avérer lourde dans le cas d'objets proposant un grand nombre de propriétés.
- Encoder l'ensemble des propriétés dans un objet **TempVar** unique pour cet objet.

Cette deuxième méthode se rapproche du mécanisme de **sérialisation** que l'on retrouve dans d'autres langages tels que **Java**. Malheureusement une telle fonctionnalité n'est pas native dans Visual Basic et va demander à être codée par le développeur en fonction de ses besoins et ne pourra par conséquent se limiter qu'aux objets simples dont les propriétés sont des données et non d'autres objets.

Ce mécanisme de pseudo-sérialisation est à réaliser en plusieurs étapes :

1. Stocker dans une chaine de caractères l'ensemble des propriétés de l'objet en séparant chacune d'elles par un (ou des) caractère(s) improbable(s).
2. Créer une méthode dans la classe de l'objet permettant de valoriser ses propriétés en fonction de la chaine issue de la sérialisation.



Comme en atteste les couleurs sur le schéma ci-dessus, les deux objets, bien qu'ils possèdent les mêmes propriétés, sont différents : la **désérialisation** a créé un nouvel objet disposant des mêmes propriétés que l'original.

Voici un exemple de classe **clsPersonne** :

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
Option Compare Database

'---------------Membres privés---------------
Private p_strNom As String
Private p_strPrenom As String
Private p_intAge As Integer

'------------------Propriétés----------------
Public Property Get Nom() As String
    Nom = p_strNom
End Property
Public Property Let Nom(strNom As String)
    p_strNom = strNom
End Property
Public Property Get Prenom() As String
    Prenom = p_strPrenom
End Property
Public Property Let Prenom(strPrenom As String)
    p_strPrenom = strPrenom
End Property
Public Property Get Age() As Integer
    Age = p_intAge
End Property
Public Property Let Age(intAge As Integer)
    p_intAge = intAge
End Property

Public Sub SePresenter()
    MsgBox "Je m'appelle " & p_strNom & " " & p_strPrenom & " et j'ai " & p_intAge & " ans"
End Sub
```

Nous allons la modifier pour qu'elle expose les méthodes permettant son clonage.

```
Public Function Serialiser() As String
    Serialiser = p_strNom & "#" & p_strPrenom & "#" & p_intAge
End Function

Public Sub Deserialiser(strChaine As String)
    'Découpe de la chaine de sérialisation
    Dim strTemp() As String
    strTemp = Split(strChaine, "#")

    'Affectation aux membres privés
    p_strNom = strTemp(0)
    p_strPrenom = strTemp(1)
    p_intAge = CInt(strTemp(2))
End Sub
```

Exemple d'utilisation :

```
Sub Exemple()
Dim oPersonne1 As New clsPersonne
With oPersonne1
    .Nom = "DUPOND"
    .Prenom = "Paul"
    .Age = 22
End With

'Ajoute la variable temporaire
Application.TempVars.Add "Ma Personne", oPersonne1.Serialiser
'Détruit l'objet oPersonne1
Set oPersonne1 = Nothing
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
'Recrée un objet identique à oPersonne1 depuis la variable temporaire
Dim oPersonne2 As New clsPersonne
oPersonne2.Deserialiser Application.TempVars("Ma Personne")

'Interroge le nouvel objet
oPersonne2.SePresenter


End Sub
```

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VII - Persistance des données

Bien qu'il s'agisse de données temporaires, il n'en reste pas moins que la question du stockage peut arriver à un moment ou un autre. En effet, si la durée de vie d'une donnée stockée dans une base se veut " éternelle ", celle d'une variable est limitée à celle de l'utilisation du projet. Le but de ce chapitre : proposer des alternatives permettant de stocker les informations ne faisant pas partie du domaine de gestion en tenant compte du fait que l'application soit ou non multi-utilisateurs.

Le domaine de gestion est un référentiel regroupant l'ensemble des données prises en compte lors de la modélisation de la base. Il peut s'agir d'un client, d'une commande, etc. Une variable d'application telle que nous l'avons vue plus haut dans cet article ne fait pas partie de ce domaine. Exemple : le nom de l'utilisateur de l'application.

VII-A - Cas des applications mono-utilisateur et mono-poste

VII-A-1 - Problématique

Si vous utilisez le nouveau format de données *accdb* et les nouveautés du moteur *ACE*, il est fort probable que vous soyez dans cette situation, à savoir : un seul fichier Access déployé sur un seul poste.

Dans ce cas, le stockage des variables temporaires pour une utilisation ultérieure (autre session du système, après un arrêt, etc) est assez simple, il suffit de disposer d'une table où sera inscrite les variables à chaque modification.

La table nommée **tbl_util_variable** possèdera deux champs de type texte :

- **VarNom** : nom de la variable
- **VarValeur** : valeur de la variable

A l'ouverture de la base de données la macro **AutoExec** aura en charge de recréer chacune des variables définies dans la table. Ceci se fera notamment à l'aide d'un recordset DAO basé sur la table **tbl_util_variable**. La seule difficulté réside dans le choix de la technologie à utiliser pour écrire les variables dans la table. En effet, il n'existe pas d'évènement exécuté à la fermeture de l'application qui pourrait permettre de modifier le contenu de la table en fonction des éléments présents dans la collection **Application.TempVars**. Il sera donc nécessaire de modifier le contenu de la table à la volée (c'est-à-dire lors de la création, modification, ou suppression d'une variable temporaire). Malheureusement, une nouvelle fois, aucun système n'a été prévu dans ce sens et les variables temporaires ne déclenchent aucun évènement. Une solution consiste à créer sa propre classe encapsulant la collection **TempVars** de l'objet **Application**. Cette classe sera nommée **clsTempVars** et tous les appels d'**Application.TempsVars** seront remplacés par des instructions destinées à un objet de type **clsTempVars**.

La structure de base de la classe **clsTempVars** est la suivante :

```
Property Get TempVars() As TempVars
    Set TempVars = Application.TempVars
End Property

Property Let Item(strVarNom As String, strVarValeur As String)
    Application.TempVars(strVarNom) = strVarValeur
End Property
Property Get Item(strVarNom As String) As String
    Item = Nz(Application.TempVars(strVarNom))
End Property
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
Sub Remove(strVarNom As String)
    Application.TempVars.Remove strVarNom
End Sub
Sub RemoveAll()
    Application.TempVars.RemoveAll
End Sub
Sub Add(strVarNom As String, strVarValeur As String)
    Me.Item(strVarNom) = strVarValeur
End Sub
```

Si avant l'affectation de la variable *Essai* à *100* était obtenu à l'aide de :

```
Application.TempVars("Essai") = 100
```

Il faut maintenant utilisé :

```
Dim oTmpVar As New clsTempVars
oTmpVar.Item("Essai") = 100
```

A première vue, l'intérêt est moindre. Cependant, les méthodes **Remove**, **Add**, et **Item** peuvent facilement être modifiées pour prendre en compte le traitement à appliquer aux enregistrements de la table **tbl_util_variable**. En comparaison avec des langages objets plus évolués (Java, **DotNet**), nous pourrions presque dire que la classe **clsTempVars** est un **héritage** de la classe **TempVars** et que ses différentes méthodes sont surchargées.

Bien entendu, du fait que les variables temporaires créées par cette classe sont ajoutées à la collection **Application.TempVars**, les instructions de consultations telles que celles ci-dessous sont toujours valides et les variables temporaires gardent une portée universelle (formulaire, requête, VBA, automation, etc.)

```
MsgBox Application.TempVars("Essai")
```

*Pour créer la classe vue précédemment, vous devez créer un nouveau module de classe, l'enregistrer sous le nom **clsTempVars** et y écrire le code VBA donné.*

VII-A-2 - Enrichissement de la classe clsTempVars pour la création des variables

Comme indiqué plus haut, au démarrage, la macro **AutoExec** sera chargée de consulter la table **tbl_util_variable** et de créer les différentes variables temporaires pour l'application. Un recordset **DAO** est nécessaire.

```
Sub Initialiser()

    'Nom de la table contenant les variables
    Const NOMTABLE = "tbl_util_variable"

    'Déclaration des variables DAO
    Dim oDb As DAO.Database
    Dim oRst As DAO.Recordset

    'Accès à la base de données
    Set oDb = CurrentDb
    'Ouverture du recordset
    Set oRst = oDb.OpenRecordset("tbl_util_variable", dbOpenForwardOnly)
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
    With oRst
        'Parcours les enregistrements jusqu'à la fin du recordset
        'et crée les variables temporaires une à une
        While Not .EOF
            Application.TempVars.Add .Fields(0).Value, .Fields(1).Value
            .MoveNext
        Wend
        'Ferme le recordset
        .Close
    End With

    'Libère les ressources
    Set oRst = Nothing
    Set oDb = Nothing

End Sub
```

Voici un exemple de fonction lancé par la macro **AutoExec** :

```
Function AutoExec()

    'Crée les variables temporaires définies dans la table tbl_util_variable
    Dim oTmpVar As New clsTempVars
    oTmpVar.Initialiser

End Function
```

VII-A-3 - Enrichissement de la classe clsTempVars pour l'écriture dans la table

Les méthodes **Item**, **Remove** et **RemoveAll** de la classe **clsTempsVars** doivent être modifiées afin que la table **tbl_util_variable** soit constamment la copie conforme de la classe **Application.TempVars**. La méthode **Add** n'a pas besoin de subir de modification puisqu'elle fait un simple appel à la méthode **Item**.

Les suppressions peuvent être obtenues à l'aide d'une requête.

```
Sub RemoveAll()
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Vide la collection TempVars
    Application.TempVars.RemoveAll

    'Vide la table
    Set oDb = CurrentDb
    oDb.Execute "DELETE FROM tbl_util_variable", dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Sub
```

```
Sub Remove(strVarNom As String)
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Supprime la variable temporaire
    Application.TempVars.Remove strVarNom

    'Vide la table
    Set oDb = CurrentDb
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
    oDb.Execute "DELETE FROM tbl_util_variable WHERE " & BuildCriteria("VarNom", dbText,
 strVarNom), dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Sub
```

La propriété **Item** est un peu plus complexe puisqu'elle peut être utilisée pour ajouter une nouvelle variable ou modifier la valeur d'une existante. De ce fait, un **recordset** est nécessaire afin d'éviter l'insertion d'un doublon. (Une solution à base de requête **Update/Insert** combinée à une gestion d'erreur aurait aussi pu être utilisée dans ce cas de figure)

```
Property Let Item(strVarNom As String, strVarValeur As String)

    'Décalaration des variables DAO
    Dim oDb As DAO.Database
    Dim oRst As DAO.Recordset

    'Modifie la variable temporaire
    Application.TempVars(strVarNom) = strVarValeur

    'Ouvre le recordset
    Set oDb = CurrentDb
    Set oRst = oDb.OpenRecordset("tbl_util_variable", dbOpenDynaset)

    With oRst
        'Recherche la variable dans la table
        .FindFirst BuildCriteria("VarNom", dbText, strVarNom)
        'Si non trouvé, alors crée l'enregistrement, sinon modifie
        If .NoMatch Then
            .AddNew
            .Fields(0).Value = strVarNom
            .Fields(1).Value = strVarValeur
            .Update
        Else
            .Edit
            .Fields(1).Value = strVarValeur
            .Update
        End If
        'Ferme le recordset
        .Close
    End With

    'Libère les ressources
    Set oRst = Nothing
    Set oDb = Nothing


End Property
```

VII-A-4 - Expiration des données

Stocker les variables temporaires est intéressant. Toutefois il pourrait s'avérer très utile de limiter cette persistance dans la durée. A l'instar des cookies internet, la mise en place d'une date d'expiration peut être envisagée. Dans un premier temps, il faut modifier la table **tbl_util_variable** afin d'y ajouter le champ **VarExpire** de type date qui définira la date au-delà de laquelle la donnée sera supprimée de la base. Ce champ pourra avoir la valeur **NULL** afin d'autoriser un stockage perpétuel.

La suppression des enregistrements dans la table peut être obtenue à l'aide d'une requête lancée depuis la classe **clsTempVars**.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
Private Sub DetruireExpire()
    'Décalaration des variables DAO
    Dim oDb As DAO.Database

    'Accès à la base de données
    Set oDb = CurrentDb
    'Détruit les enregistrements expirés
    oDb.Execute "DELETE FROM tbl_util_variable WHERE VarExpire<Now()", dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Sub
```

Afin d'exécuter la procédure ci-dessus à chaque utilisation des variables temporaires, elle doit être invoquée dans le constructeur (**Class_Initialize**) de la classe.

```
Private Sub Class_Initialize()
    Call DetruireExpire
End Sub
```

La définition de la date d'expiration sera définie à l'aide de la méthode **Expire**.

```
Property Let Expire(strVarNom As String, dtVarExpire As Variant)
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Met à jour la date d'expiration
    Set oDb = CurrentDb
    oDb.Execute "UPDATE tbl_util_variable SET VarExpire=" & _
                IIf(IsNull(dtVarExpire), "NULL", "#" & Format(dtVarExpire, "mm/dd/yyyy hh:nn:ss") &
 "#") & _
                " WHERE " & BuildCriteria("VarNom", dbText, strVarNom), dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Property
```

Exemple d'utilisation :

```
Dim oTmpVar As New clsTempVars
oTmpVar.Item("Essai") = 100
oTmpVar.Expire("Essai") = Now() + 20
```

VII-B - Cas des applications mono-utilisateur et multi-poste

VII-B-1 - Problématique

L'appellation est un peu mal choisie, mais il s'agit des applications où aucun compte utilisateur n'a été créé. Tous les postes utilisent le même utilisateur. La base de données dorsale est située sur un serveur, la base de données frontale est située sur chaque poste client. Un utilisateur physique est donc uniquement défini par son unité centrale depuis laquelle est lancée l'application. Afin de stocker ses variables temporaires sans qu'elles soient accessibles des autres utilisateurs il suffit d'utiliser le même mécanisme que vu précédemment pour les applications monoposte en prenant soin de stocker la table **tbl_util_variable** dans la base de données frontale (pour rappel, il s'agit de celle présente sur chaque poste client)

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VII-C - Cas des applications multi-utilisateurs et mono-poste

VII-C-1 - Problématique

Cette fois ci, la base de données est composée en général d'un seul fichier et un système multi-utilisateurs a été mis en place. Cela peut être la sécurité de niveau utilisateur de Microsoft Access dans le cadre d'un fichier mdb ou bien encore une authentification de votre propre cru (via un table d'utilisateur notamment). Contrairement au cas de figure précédent, les utilisateurs ne sont pas identifiés par leur poste mais par leur nom ou leur identifiant. Ce paramètre sera à stocker dans la table **tbl_util_variable** afin de réattribuer les variables au bon utilisateur.

VII-C-2 - Modification de la table tbl_util_variable

Jusque là, la table est structurée de la sorte :

- **VarNom** (clé primaire) : nom de la variable
- **VarValeur** : valeur de la variable
- **VarExpire** : date d'expiration de la sauvegarde

Il faut désormais stocker le nom de l'utilisateur et il est évident que plusieurs utilisateurs peuvent avoir des variables de même nom avec (ou non) des valeurs différentes.

- **VarNom** : nom de la variable
- **VarUtilisateur** : utilisateur de la variable **VarNom**
- **VarValeur** : valeur de la variable
- **VarExpire** : date d'expiration de la sauvegarde

Le couple de champs **VarNom** et **VarUtilisateur** devient la clé primaire de la table.

VII-C-3 - Modification de la classe

Nous allons partir du principe que la solution retenue est celle de la sécurité intégrée de Microsoft Access. De ce fait, la propriété **Application.CurrentUser** retourne le nom de l'utilisateur courant.

Voici les modifications à apporter à la propriété **Item** :

```
Property Let Item(strVarNom As String, strVarValeur As String)

    'Décalaration des variables DAO
    Dim oDb As DAO.Database
    Dim oRst As DAO.Recordset

    'Modifie la variable temporaire
    Application.TempVars(strVarNom) = strVarValeur

    'Ouvre le recordset
    Set oDb = CurrentDb
    Set oRst = oDb.OpenRecordset("tbl_util_variable", dbOpenDynaset)

    With oRst
        'Recherche la variable dans la table
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
        .FindFirst BuildCriteria("VarNom", dbText, strVarNom)
        'Si non trouvé, alors crée l'enregistrement, sinon modifie
        If .NoMatch Then
            .AddNew
            .Fields("VarNom").Value = strVarNom
            .Fields("VarValeur").Value = strVarValeur
            .Fields("VarUtilisateur").Value = Application.CurrentUser
            .Update
        Else
            .Edit
            .Fields(1).Value = strVarValeur
            .Update
        End If
        'Ferme le recordset
        .Close
    End With

    'Libère les ressources
    Set oRst = Nothing
    Set oDb = Nothing


End Property
Property Get Item(strVarNom As String) As String
    Item = Nz(Application.TempVars(strVarNom))
End Property
```

Code de la procédure **Initialiser**

```
Sub Initialiser()


    'Déclaration des variables DAO
    Dim oDb As DAO.Database
    Dim oRst As DAO.Recordset

    'Accès à la base de données
    Set oDb = CurrentDb
    'Ouverture du recordset
    Set oRst = oDb.OpenRecordset("SELECT * FROM tbl_util_variable WHERE " & _
                                 BuildCriteria("VarUtilisateur", dbText, Application.CurrentUser), _
                                 dbOpenForwardOnly)

    With oRst
        'Parcours les enregistrements jusqu'à la fin du recordset
        'et crée les variables temporaires une à une
        While Not .EOF
            Application.TempVars.Add .Fields(0).Value, .Fields(1).Value
            .MoveNext
        Wend
        'Ferme le recordset
        .Close
    End With

    'Libère les ressources
    Set oRst = Nothing
    Set oDb = Nothing

End Sub
```

Méthodes **Remove** et **RemoveAll** :

```
Sub Remove(strVarNom As String)
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

```
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Supprime la variable temporaire
    Application.TempVars.Remove strVarNom

    'Vide la table
    Set oDb = CurrentDb
    oDb.Execute "DELETE FROM tbl_util_variable WHERE " & BuildCriteria("VarNom", dbText, strVarNom)
 & _
                " AND " & BuildCriteria("VarUtilisateur", dbText, Application.CurrentUser), _
                dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Sub
Sub RemoveAll()
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Vide la collection TempVars
    Application.TempVars.RemoveAll

    'Vide la table
    Set oDb = CurrentDb
    oDb.Execute "DELETE FROM tbl_util_variable WHERE " & _
                BuildCriteria("VarUtilisateur", dbText, Application.CurrentUser), _
                dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Sub
```

Propriété **Expire** :

```
Property Let Expire(strVarNom As String, dtVarExpire As Variant)
    'Déclaration DAO
    Dim oDb As DAO.Database

    'Met à jour la date d'expiration
    Set oDb = CurrentDb
    oDb.Execute "UPDATE tbl_util_variable SET VarExpire=" & _
                IIf(IsNull(dtVarExpire), "NULL", "#" & Format(dtVarExpire, "mm/dd/yyyy hh:nn:ss") &
 "#") & _
                " WHERE " & BuildCriteria("VarNom", dbText, strVarNom) & _
                " AND " & BuildCriteria("VarUtilisateur", dbText, Application.CurrentUser), _
                dbFailOnError

    'Libère les ressources
    Set oDb = Nothing
End Property
```

Notez que la procédure **DetruireExpire** n'a pas besoin d'être modifiée. Cela permettra de supprimer les enregistrements expirés des utilisateurs qui ne se connectent pas à la base de données.

L'utilisation de la classe quant à elle n'a pas changé :

```
Dim oTmpVar As New clsTempVars
oTmpVar.Item("Essai") = 100
oTmpVar.Expire("Essai") = Now() + 20
```


Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VII-D - Cas des applications multi-utilisateurs et multi-postes

VII-D-1 - Problématique

Il s'agit des applications les plus complexes, les utilisateurs doivent avoir accès à leurs variables sauvegardées dans la table **tbl_util_variable** depuis n'importe quel poste. De ce fait la table doit être stockée dans la base de données dorsale et la classe **clsTempVars** sera la même que celle décrite juste avant.

VII-D-2 - Données partagées

Jusqu'ici nous avons traité uniquement les variables propres à chaque utilisateur. Mais qu'en est-il des données devant être partagées entre les différents utilisateurs. Avec les variables temporaires, chaque poste client héberge une version des données présentes dans la table **tbl_util_variable** du serveur. Que va-t-il se passer si un poste client modifie une de ces variables ? La table **tbl_util_variable** sera mise à jour mais aucun des autres clients n'en sera averti. Ils disposeront donc d'une version périmée des variables temporaires. Il est donc difficile voire dangereux d'utiliser les variables temporaires dans une telle situation. A chaque fois qu'une donnée partagée est nécessaire, elle doit impérativement être lue directement depuis le serveur afin d'éviter tout risque de péremption.

En résumé

Le module de classe donné ici en exemple permet la sauvegarde des variables temporaires en fonction de l'endroit où sera stockée la table **tbl_util_variable** :

	Mono utilisateur	**Multi utilisateur**
Mono poste	Base de données frontale	Base de données frontale
Multi postes	Base de données frontale	Base de données dorsale

Dans le cas d'application multi-postes, il est impossible de partager les mêmes variables temporaires sans prendre le risque d'avoir des versions périmées de ces dernières pour certains clients, étant donné qu'aucun moyen n'est mis en #uvre pour avertir les autres postes lorsque l'un d'entre eux procède à une modification.

Dans le cas d'une application mono utilisateur et mono poste, il est possible d'utiliser les objets ***DAO.Property*** *en remplacement de la table* ***tbl_util_variable****.*

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

VIII - Conclusion

A travers ce document nous avons fait le tour des possibilités offertes par les variables temporaires des applications Office 2007. Dans bien des cas, elles vous seront utiles mais retenez-en les avantages et inconvénients qui ont été abordés jusque-là :

Avantages :

- Il s'agit d'une fonctionnalité permettent des échanges succincts d'information via automation.
- Leur regroupement au sein de la collection **TempVars** permet aux développeurs friands de variables globales de mieux structurer leur code. En parcourant la collection en fin de projet, le développeur peut facilement déterminer les variables inutilisées ou redontantes.
- Leur nombre est suffisant pour pallier à toutes les alternatives.

Inconvénients :

- De type **Variant**, elles occupent parfois de la mémoire inutilement.
- Etant données qu'elles sont accessibles depuis l'extérieur, des applications malveillantes peuvent tenter de les manipuler.
- Il aurait été intéressant que l'application lève un évènement lors de la modification, de l'ajout et de la suppression d'une variable, permettant ainsi une détection des accès intrusifs.

En résumé, j'aurais tendance à présenter les variables temporaires de la sorte :

Un bon compromis entre les partisans et les opposants aux variables globales offrant une organisation structurée du code mais souffrant toujours d'une allocation souvent trop importante de mémoire vive.

Je tiens à remercier l'ensemble des membres de l'équipe Access ainsi que Dut de developpez.com pour leur précieuse relecture.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.

Ce document est issu de http://www.developpez.com et reste la propriété exclusive de son auteur. La copie, modification et/ou distribution par quelque moyen que ce soit est soumise à l'obtention préalable de l'autorisation de l'auteur.