Règles-Métier et Intégrité Référentielle : comment traiter côté client les règles déclarées côté serveur…

Avec VFP, j'ai l'habitude après un TABLEUPDATE() sur une vue distante, de traiter son échec en analysant le AERROR rempli par l'ODBC (principalement les lignes 3 et 5 de ce tableau, qui me donnent le texte et le n° de cette erreur ODBC).
Cette démarche permet de coder les contraintes d'IR une seule fois (coté serveur), elle facilite la maintenance.
Comment faire aujourd'hui, avec un client développé en .NET ?

Dans cette brève étude, je vais montrer comment utiliser quelques unes des fonctionnalités de StrataFrame pour récupérer les règles d'Intégrité Référentielles existant côté serveur. Nous traiterons ces règles aussi bien dans la couche métier que dans la couche de présentation.

Cet exemple s'appuiera sur la gestion de la TVA dans une application réelle de gestion commerciale (LOGICAVEAU). Les données sont sur un serveur SQL Server 2005, le client est développé en VB.NET avec StrataFrame 1.6.6

Notre exemple côté serveur

Dans toutes mes bases de données, j'utilise systématiquement les PK sur des clés déléguées (champs Integer Identity). Il y a donc une clé unique sur un autre champ qui est présenté à l'interface utilisateur.

Dans cette application, les tables appartiennent au schéma dbo, qui n'est PAS accessible aux applications clientes : elles se connectent en authentification SQL Server sur le schéma LogicaveauApp, qui ne contient que des vues, des procédures stockées, des fonctions.

Quelle Règles pour la TVA ?

L'IR pour cette table TVA doit obliger à respecter les règles suivantes :

- Le champ CODE_TVA doit être unique
- On ne peut pas supprimer d'enregistrement de cette table si celui-ci est utilisé dans un enregistrement de la table ARTICLE
- On ne peut pas supprimer d'enregistrement de cette table s'il est le seul à ce taux, et que ce taux fait partie des taux présents dans la table TVABASIQUE
- On ne peut modifier le taux d'un enregistrement de cette table s'il est le seul à ce taux, et que ce taux fait partie des taux présents dans la table TVABASIQUE (on doit d'abord passer par la table TVABASIQUE)
- Si le champ TAUX d'un enregistrement a une valeur différente de 0, on ne peut pas le mettre à jour à 0

Les 2 premières règles sont des règles d'IR (Intégrité Référentielles) que nous retrouverons de façon identique dans de nombreuses tables ; les 3 suivantes sont plutôt des Règles-Métier, bien spécifiques à cette table.

IR par Contraintes, IR par Triggers

Contraintes

La première règle d'IR est obtenue de façon structurelle, en utilisant la syntaxe CONSTRAINT :

```
ALTER TABLE [dbo].[articles]  ADD  CONSTRAINT [fk_tva_fk] FOREIGN KEY([tva_fk]) REFERENCES
[dbo].[tva] ([tva_pk])
```

La violation de cette règle déclenche une erreur qui annule l'ordre SQL reçu et fait remonter un message d'erreur par l'ODBC vers le client.

C'est la technique préconisée par Microsoft pour SL Server 2005, comme étant la plus rapide. Le nouvel assistant upsizing VFP permet de choisir cette approche lors d'une migration des données FoxPro vers SQL Server 2005.

triggers

Les autres règles sont traitées par des triggers, qui seront exécutés après que la contrainte ait été vérifiée.

Le trigger DELETE contient essentiellement les lignes suivantes :

```
SELECT @combien= COUNT(*) FROM deleted, dbo.tva WHERE (tva.[taux] = deleted.[taux])
IF @combien=0 and (SELECT COUNT(*) FROM deleted, typetvabasique WHERE (deleted.[taux] =
typetvabasique.[taux_fk])) > 0
```

(pour ceux et celles qui ne sont pas familiers de SQL Server, je rappelle que SQL Server nous présente les enregistrements faisant l'objet d'un DELETE dans une table temporaire DELETED accessible le temps de la transaction ; lors d'un UPDATE, on utilise les tables temporaires INSERTED et DELETED)

Si ces conditions de violation d'IR sont réunies, l'erreur 44445 est levée, un ROLLBACK est exécuté.

Coté client, maintenant…

Rappelons que StrataFrame nous propose un ensemble structuré de classes pour une architecture n-layers, depuis la connexion jusqu'à la couche de présentation. Au centre de cette architecture se trouve la classe du Business Object (BO), qui communique d'un côté avec la couche d'accès (classes DataLayer et DBDataSourceItem), et de l'autre avec tous les contrôles du framework .net surchargés par StrataFrame.

Dans mes développements, j'utilise uniquement des classes dérivées de celles proposées par StrataFrame, et les BO d'une application sont compilés dans une DLL référencée dans le projet UI.

Intégrité Référentielle

Comment ajouter un nouvel enregistrement avec StrataFrame

NewRow et Add

Il y a 3 façons d'ajouter un nouvel enregistrement, avec StrataFrame.

- La méthode *NewRow()* du Business Object crée un nouvel enregistrement dans la DataTable du BO, et va se positionner sur ce nouvel enregistrement. L'état d'édition n'est pas modifié, aucun contrôle lié ne sera rafraichi ou mis à jour. C'est la méthode privilégiée en codage non-visuel
- La méthode *Add()*du Business Object fait de même, mais en plus, elle met l'état d'édition à *Adding*, elle déclenche l'événement *EditingStateChanged*, et elle rafraichit les contrôles liés.

- La méthode *Add()* du Form appelle la méthode *Add()* du BO déclaré en *PrimaryBusinessObject* sur ce Form.

Ce sont les méthodes *Save()* qui vont transmettre ce nouvel enregistrement vers le SGBDR. On trouve une méthode *Save()* sur le Business Object, et une sur le Form. Ces 2 méthodes diffèrent par les séquences d'événements qu'elles génèrent :

Save() depuis un Business Object

1. *CheckRulesOnCurrentRow* Event
2. *BusinessRulesChecked* Event
3. *BeforeSave* Event
4. *AfterSave* Event
5. *EditingStateChanged* Event

Voir le diagramme complet Annexes

Save depuis le BO

Save() depuis un Form

1. *CheckRulesOnCurrentRow* Event (Business Objects)
2. *BusinessRulesChecked* Event (Business Objects)
3. *BusinessRulesChecked* Event (Form)
4. *BeforeSave* Event (Business Objects)
5. *BeforeSave* Event (Form)
6. *AfterSave* Event (Business Objects)
7. *AfterSave* Event (Form)
8. *EditingStateChanged* Event (Business Objects)
9. *EditingStateChanged* Event (Form)

Voir le diagramme complet Save depuis le Form

Les BusinessRules

Dans leur forme minimale (en utilisant l'assistant proposé), elles garantissent que les champs obligatoires ont été saisis. Elles sont définies sur le BO, et remontent les éventuels messages d'erreurs vers l'interface utilisateur. Elles assurent les règles-métiers essentielles, et ne me semblent pas destinées à vérifier l'intégrité référentielle.

Les méthodes Before

La classe SF StandardForm propose les méthodes *BeforeDelete* et *BeforeSave*, qui peuvent parfaitement être utilisée pour une approche qui anticipe l'erreur.

Une fonction sur le BO

Nous devons coder une fonction sur le Business Object, qui vérifie si le code TVA passé en paramètre existe déjà. Par exemple :

```
Public Function verif_Code_existe(ByVal code_tva As String) As Boolean
    Dim loCmdSQL As New SqlCommand
    loCmdSQL.CommandText = "select count(*) from v_tva where code_tva = @code_tva"
    loCmdSQL.Parameters.Add("@code_tva", SqlDbType.Char)
    loCmdSQL.Parameters("@code_tva").Value = code_tva
    Return CBool(CInt(Me.ExecuteScalar(loCmdSQL)))
```

```
End Function
```

Un petit peu de code dans le Form

La méthode BeforeSave du Form contiendra alors :

```
Private Sub frmTVA_BeforeSave(ByVal e As
MicroFour.StrataFrame.Data.BeforeSaveUndoByFormEventArgs) _
Handles MyBase.BeforeSave
    If Me.SFBOTVA.verif_Code_existe(Me.txtCode_TVA.Text) = True Then
        e.Cancel = True
        MsgBox("Ce code existe déjà") 'ce message sera modifé ensuite
    End If
End Sub
```

Cette technique « par anticipation » fonctionne très bien, et se prête à un codage rapide mais peu facile à maintenir. Il faut en effet être certain que tous les BO disposent d'une fonction de vérification, et il faut modifier ces fonctions en cas de changement structurel côté serveur.
Afin de faciliter la maintenance, j'utiliserai les méthodes Before du Business Object plutôt que celles du Form, en renvoyant le résultat vers la couche visuelle pour un éventuel traitement.

Le traitement des exceptions

La démarche par traitement des exceptions consiste à envoyer les commandes Save ou Delete sur le serveur, et à récupérer et traiter l'erreur retournée si nécessaire. Elle ressemble au traitement par analyse du retour de TableUpdate() en VFP. C'est d'ailleurs la raison pour laquelle je privilégie cette démarche. C'est le traitement privilégié de l'intégrité référentielle.

Nous allons tout coder dans les classes, aucune chaine de caractère constituant les message explicatifs n'y figurera (nous utiliserons le module de Messaging de StrataFrame).

SAVE

Nous disposons de 2 techniques pour traiter les exceptions conséquentes d'un *Save()*, que nous choisissons par le réglage de la propriété *ErrorSavingMode* du Business Object : si cette propriété est à *ContinueOnError*, l'exception est transférée à l'évènement *ErrorSaving*, si elle est à *FailOnError*, nous devons poser un Try/Catch dans le Save lui-même.L'exemple ci-dessous présente un traitement par *FailOnError*, le même code pourrait se trouver dans le *ErrorSaving*.
Save est en fait une fonction du Form, qui exécute le Save proprement dit, et qui renvoie le résultat Integer dans l'énumération *SaveUndoResult*. Cette fonction étant déclarée Overridable dans la classe Standard Form de StrataFrame, il nous suffit de coder notre fonction dans notre classe dérivée de celle-ci :

```
Public Overrides Function save() As SaveUndoResult
    Try
        MyBase.Save()

    Catch ex As Exception
        Dim MessageTexte As String = String.Empty
        MessageTexte = Me.construit_message(ex, CRUD_Errors.Create)
        MessageForm.ShowMessageByKey("NoCreate", MessageTexte)
        MyBase.Undo()
    End Try

End Function
```

S'il y a violation de la contrainte d'unicité, le Save de la structure Try va déclencher une erreur, qui sera réceptionnée dans le Catch.

Le code du Catch lance alors la construction d'un texte, par l'appel de la fonction Construit_Message, qui est définie également sur cette classe dérivée (et de laquelle seront dérivés tous nos Forms. Cette fonction reçoit elle aussi l'objet exception en paramètre, ainsi que l'origine de cette exception (j'ai défini dans cette classe une énumération CRUD_Errors, dont un membre est Create).

La fonction Construit_Message contient ce code, pour traiter le cas du Create :

```
Private Function construit_message(ByVal ex As Exception, ByVal CRUD As CRUD_Errors) As String
    Dim MessageRetour As String = String.Empty
    Dim ErrorMessage As String = String.Empty

    Select Case ex.GetType.Name.ToUpper

... ... ...

        Case "BUSINESSLAYEREXCEPTION"
            ErrorMessage = ex.InnerException.Message
            Select Case CRUD
                Case CRUD_Errors.Create
                    If ErrorMessage.Contains("UNIQUE KEY") Then
                        If Not Me.FocusControlOnAdd Is Nothing Then
                            MessageRetour = _
                            Me.FocusControlOnAdd.Text.Trim & _
                            Localization.RetrieveTextValue("NoCreateViolationRI", ErrorMessage)
                        Else
                            If Not Me.FocusControlOnEdit Is Nothing Then
                                MessageRetour = _
                                Me.FocusControlOnEdit.Text.Trim & _
Localization.RetrieveTextValue("NoCreateViolationRI", ErrorMessage)
                            Else
                                MessageRetour =
Localization.RetrieveTextValue("NoCreateGenerique", ErrorMessage)
                            End If
                        End If

                    Else
                        MessageRetour = Localization.RetrieveTextValue("NoCreateGenerique",
ErrorMessage)
                    End If

            End Select
        End Select 'ex.GetType.Name.ToUpper

        Return MessageRetour
    End Function
```

La violation d'une contrainte RI sur le serveur renvoie une erreur de type BusinessLayerException (cette information est facile à trouver en mettant le BO à *ErrorSavingMode=ContinueOnError*, ou bien en lisant le contenu du ApplicationErrorWindow que StrataFrame présente en cas d'exception non gérée (voir ApplicationErrorWindow).

Le message renvoyé par le serveur est le suivant :

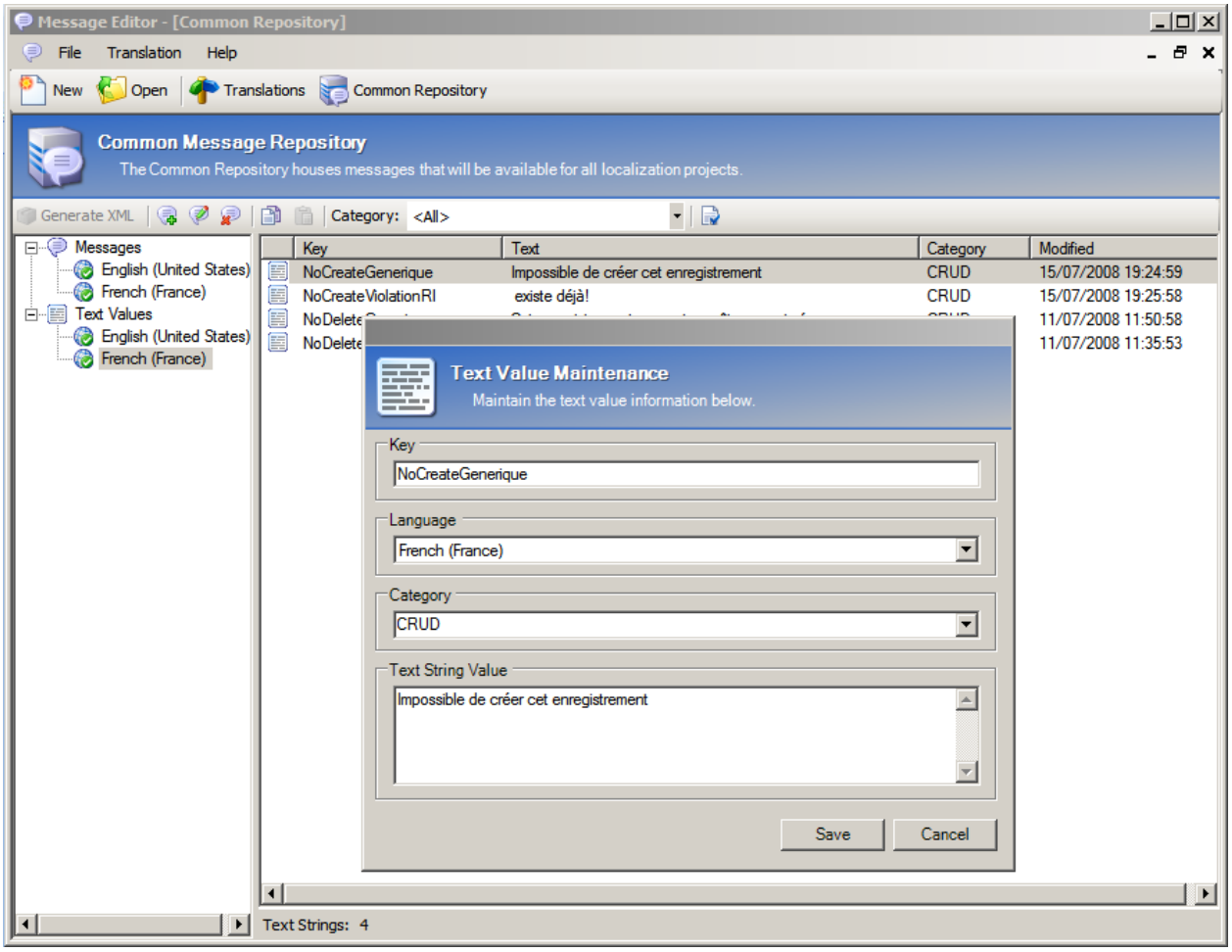
```
Violation de la contrainte UNIQUE KEY 'UQ_tva'. Impossible d'insérer une clé en double dans l'objet 'dbo.tva'.
```

C'est typiquement un message système, qu'on ne peut pas présenter à un utilisateur final en réponse à une erreur de saisie involontaire ! Construisons nous-mêmes notre message, mais de façon générique dans la classe.

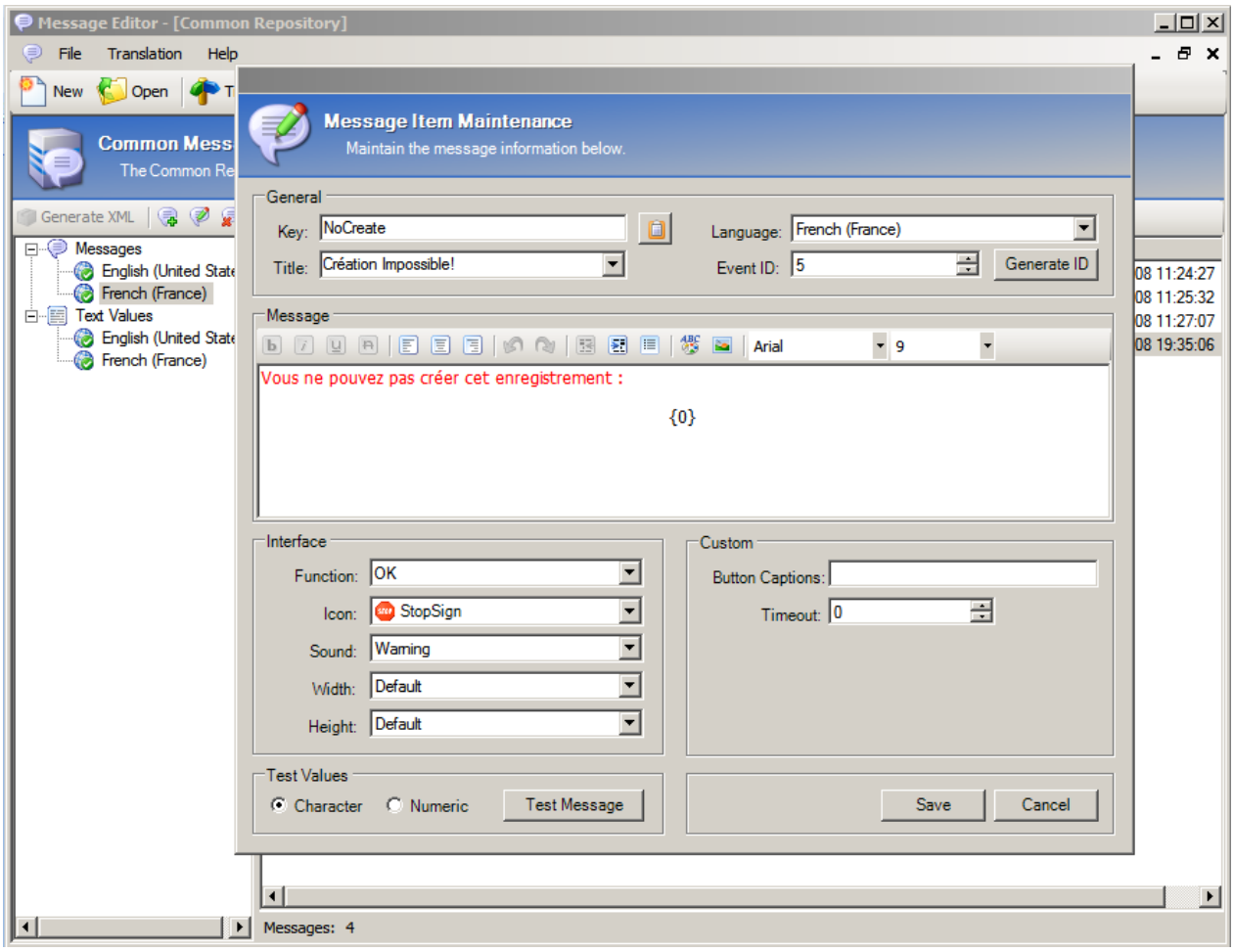
Les forms StrataFrame nous proposent d'identifier le contrôle qui recevra le focus en cas d'ajout de nouvel enregistrement (méthode *Add*) et celui que le recevra en cas de modification d'enregistrement (méthode *Edit*). Ces 2 contrôles sont identifiés dans les propriétés *FocusControlOnAdd* et *FocusControlOnEdit*. Notre fonction de création de message va donc récupérer le texte du *FocusControlOnAdd* s'il est renseigné, ou bien celui du *FocusControlOnEdit*.

C’est maintenant qu’intervient le module de « Messaging » de StrataFrame. Ce module permet de stocker dans une base de données SQL Server dédiée tous les textes qui peuvent être présentés à l’utilisateur final (labels, titres, textes de messages, etc…). Une interface conviviale permet la gestion de ces textes, leur extraction dans un fichier xml qui sera compilé dans l’application si nécessaire, et bien d’autres fonctionnalités facilitant la gestion multilingue.

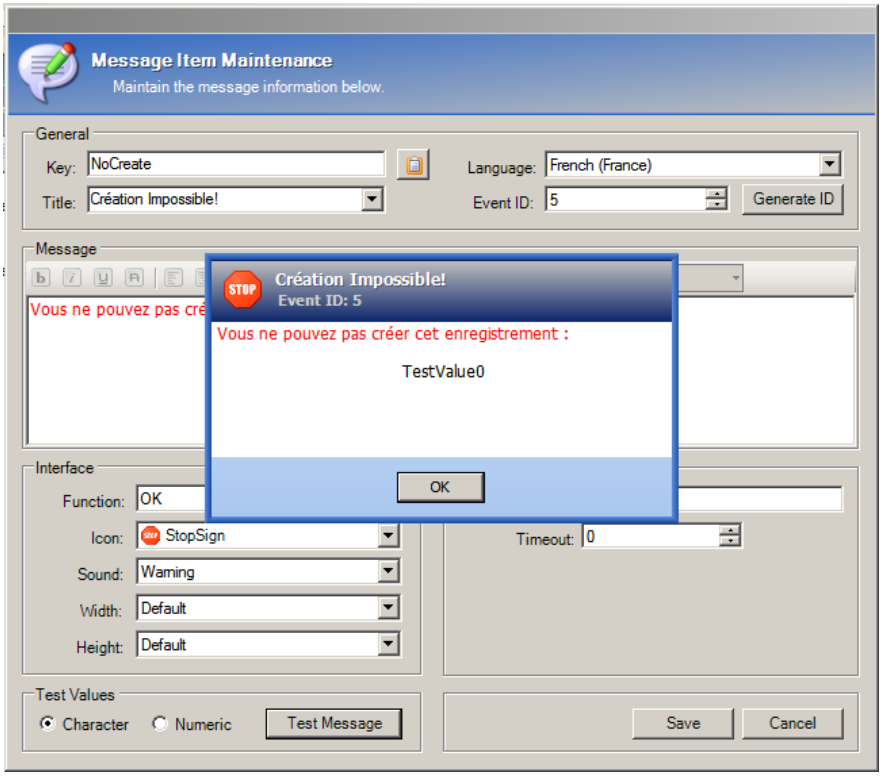
Le message NoCreateGenerique sera donc utilisé s’il n’est pas possible de concaténer le contenu d’un des contrôles spécifiques avec la chaine du message NoCreateViolationRI.



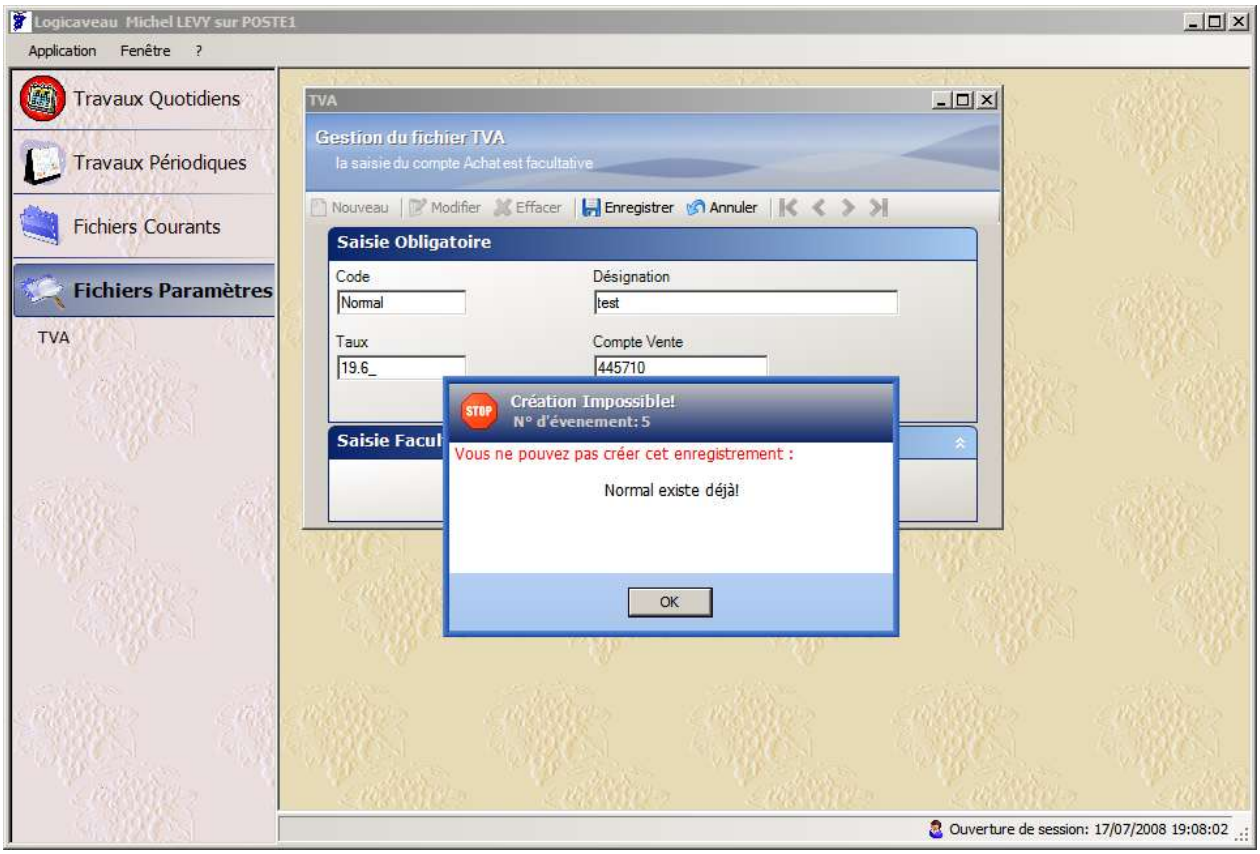
Notre message est maintenant construit, nous le mettons en forme. Plutôt qu’un MessageBox, nous utiliserons la classe StrataFrame *MessageForm*, en intégrant le messaging. Cette classe est beaucoup plus riche fonctionnellement, et sensible au thème graphique choisi par l’utilisateur



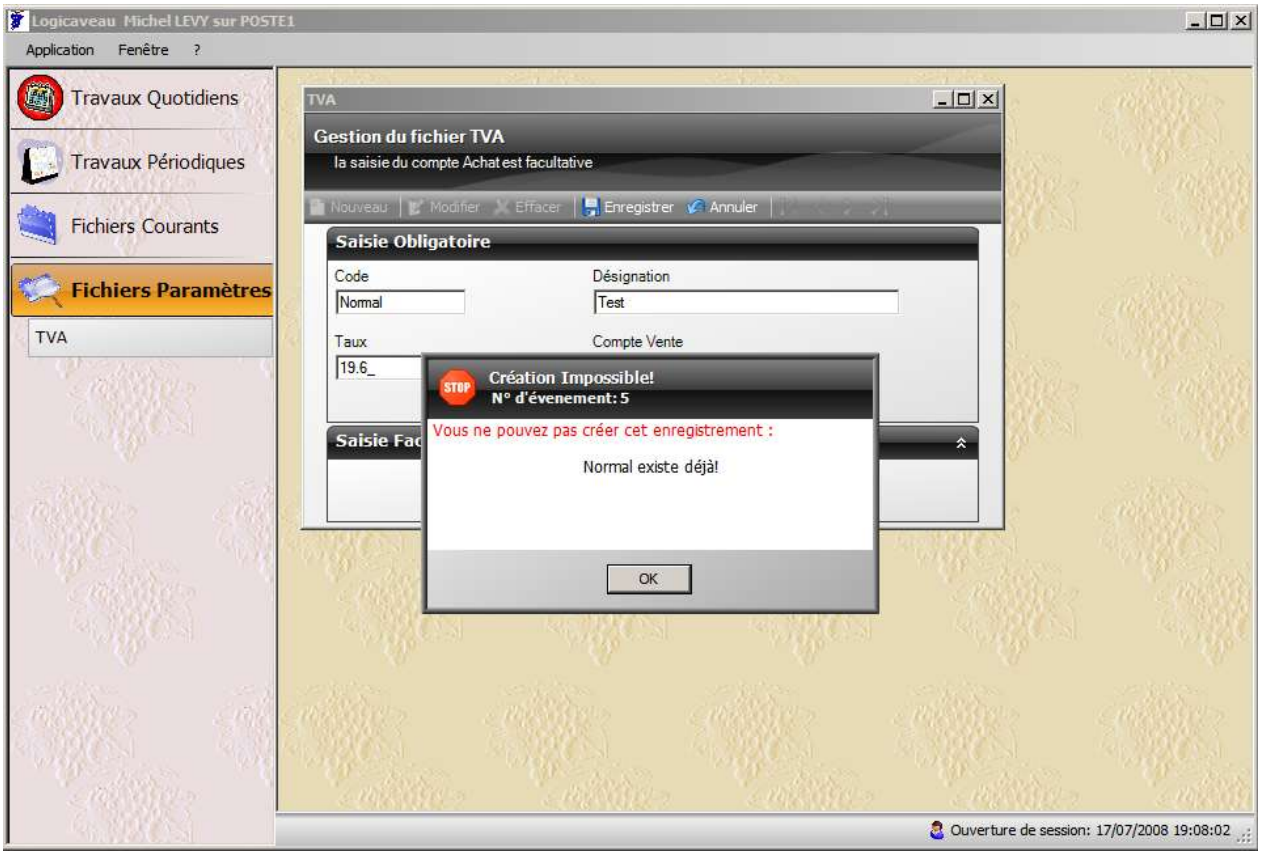
Testons le rendu de notre message :



Et le message réel est donc :



Et dans un thème graphique différent :



Le MessageForm que nous avons instancié ne comporte que le seul bouton OK, nous n'analyserons donc pas sa valeur de retour.

La dernière ligne de notre traitement est simplement un appel à la méthode *Undo()* de la classe de base. Cette méthode est semblable au TableRevert de FoxPro, elle annule les modifications faites dans la DataTable du Business Object. Il n'est pas obligatoire d'appeler le Undo, c'est seulement un choix ergonomique : on peut parfaitement préférer revenir en mode de saisie pour permettre à l'utilisateur de corriger son erreur.

Comment supprimer un nouvel enregistrement avec StrataFrame

La suppression d'un enregistrement avec StrataFrame peut se faire depuis le Business Object par 2 méthodes : *DeleteByPrimaryKey()* ou *DeleteCurrentRow()*.
DeleteByPrimaryKey nécessite une PK en paramètre, et ne sera donc pas utilisée depuis un Form (elle ne déclenche d'ailleurs pas les évènements Before et After). La méthode Delete du Form appelle un *DeleteCurrentRow*.
DeleteCurrentRow sans paramètre exécute immédiatement la commande sur le serveur, sans nécessiter de Save. On peut lui passer le paramètre *OnlyMarkAsDeleted* à Vrai ; il faudra alors un Save pour exécuter la commande sur le serveur.

Ayant choisi de privilégier le traitement des exceptions, j'utilise la version sans paramètre (plus exactement avec les 2 paramètres possibles à Faux, ce qui est fonctionnellement identique.

Ma classe de Form possède donc le code suivant :

```
Public Overrides Sub Delete(ByVal OnlyMarkAsDeleted As Boolean, ByVal CheckSecurity As Boolean)
    Try
        MyBase.Delete(OnlyMarkAsDeleted, CheckSecurity)

    Catch ex As Exception
        Dim extype As Type
        extype = ex.GetType
        Me._ErrorDeleting(ex)

    End Try
End Sub
```

La classe de Form de StrataFrame ne propose pas de méthode ErrorDeleting ni d'évènement du même nom ; je les ai donc simplement créé.

```
Public Event ErrorDeleting(ByVal ex As Exception)


Public Overridable Sub _ErrorDeleting(ByVal ex As Exception) Handles Me.ErrorDeleting
    Dim MessageTexte As String = String.Empty
    MessageTexte = Me.construit_message(ex, CRUD_Errors.Delete)
    MessageForm.ShowMessageByKey("NoDelete", MessageTexte)
End Sub
```

La fonction Construit_Message traitera donc cette exception de *Delete* sur le même principe que pour le *Create* (identification du message retourné par le serveur, construction de la chaine de texte explicite) :

```
Case CRUD_Errors.Delete
    If ErrorMessage.Contains("REFERENCE") Then
        If Not Me.FocusControlOnAdd Is Nothing Then
            MessageRetour = _
            Me.FocusControlOnAdd.Text.Trim & _
            Localization.RetrieveTextValue("NoDeleteViolationRI", ErrorMessage)
        Else
            If Not Me.FocusControlOnEdit Is Nothing Then
                MessageRetour = _
                Me.FocusControlOnEdit.Text.Trim & _
                Localization.RetrieveTextValue("NoDeleteViolationRI", ErrorMessage)
            Else
                MessageRetour = Localization.RetrieveTextValue("NoDeleteGenerique",
ErrorMessage)
            End If
        End If

    Else
        MessageRetour = Localization.RetrieveTextValue("NoDeleteGenerique", ErrorMessage)
    End If
```

Et nous afficherons le message suivant, en cas de demande de suppression violant une contrainte RI :

Règles-Métier

Sur une suppression d'enregistrement

Rappelons la règle : On ne peut pas supprimer d'enregistrement de cette table s'il est le seul à ce taux, et que ce taux fait partie des taux présents dans la table TVABASIQUE.

Il s'agit bien d'une règle-métier, qui ne s'applique qu'à cette table (en fait, toutes mes tables de paramétrage sont adossées à de telles tables basiques afin de garantir une réelle Intégrité des données).

Cette règle doit être vérifiée au niveau de la classe de BO v_TVA (le Business Object qui pointe sur une vue sur la table TVA), et doit pouvoir être vérifiée depuis le form « Gestion du fichier TVA ».
Comme je n'ai qu'un seul BO sur ce Form, je vais donc coder tout simplement dans le *BeforeDelete* de cette instance de BO plutôt que dans celui du Form.

```
Private Sub V_TVA_BeforeDelete(ByVal e As _
MicroFour.StrataFrame.Business.BeforeDeleteEventArgs) _
Handles V_TVA.BeforeDelete

If e.Cancel = True Then
        MessageForm.ShowMessageByKey("NoDelete", _
        "Il s'agit d'un taux primaire," & vbCrLf & _
        "et c'est le seul enregistrement à ce taux")
    End If

End Sub
```

C'est donc le *BeforeDelete* codé au niveau de la classe de BO v_TVA qui vérifie la règle, la couche visuelle du Form assurant uniquement l'affichage du MessageForm correspondant.

Dans la classe du BO v_TVA, je vais donc coder :

```
Private Sub v_TVA_BeforeDelete(ByVal e As _
MicroFour.StrataFrame.Business.BeforeDeleteEventArgs) _
Handles MyBase.BeforeDelete

    Dim filtre_old As String
    With Me
        filtre_old = .Filter
        .Filter = "taux = " & .taux.ToString
        If .TauxTVAestPrimaire(.taux) AndAlso .Count = 1 Then
            e.Cancel = True
        End If
        .ParentForm.AutoShowDeleteConfirmation = Not e.Cancel
        .Filter = filtre_old
    End With

End Sub
```

Remarquons au passage l'utilisation de la propriété Filter de la classe BO… La fonction TauxTVAestPrimaire est :

```
Public Function TauxTVAestPrimaire(ByVal tauxTVA As Decimal) As Boolean

    Dim loCmdSQL As New SqlCommand
    loCmdSQL.CommandText = "" & _
    "SELECT dbo.TauxTVAestPrimaire(@taux)"
    loCmdSQL.Parameters.Add("@taux", SqlDbType.Decimal)
    loCmdSQL.Parameters("@taux").Value = tauxTVA

    Return CType(Me.ExecuteScalar(loCmdSQL), Boolean)

End Function
```

On voit qu'elle ne fait qu'exécuter une fonction sur le serveur SQL : en effet, la connexion utilisée par l'application ne lui donne pas le droit de requêter directement les tables basiques des paramètres (pas même leurs vues), mais

seulement de passer par certaines fonctions et procédures stockées.
Si le résultat est vrai et que le filtre appliqué indique que je n'ai que cet enregistrement à ce taux, je provoque l'annulation du Delete en forçant sa propriété Cancel à Vrai. Je n'ai donc plus besoin de la demande automatique de confirmation de suppression, je peux donc ajuster le *AutoShowDeleteConfirmation* du Form hôte à faux.

Sur un ajout ou une modification d'enregistrement

Notre objectif sera ici de nous assurer que les règles-métier sont bien respectées, et de permettre à l'utilisateur de modifier sa saisie si au moins une de ces règles est violée.

Nous utiliserons alors *CheckRulesOnCurrentRow*, c'est le 1er évènement du BO déclenché par une demande de *Save()*. Cet évènement est déclenché pour chaque ligne de la *DataTable* du BO ; si une ou plusieurs règles-métier sont violées, elles seront signalées dans l'interface de saisie, et le résultat du Save ne sera plus *Success*, mais *AbortedWithBrokenRules* (ou *CompletedWithExceptions* si nous forçons quand même ce Save).

Soit nous codons le message de cette règle en dur, et alors nous utilisons la méthode *AddBrokenRule*, soit nous voulons récupérer une chaine stockée dans le module de Messaging, et nous utilisons la méthode *AddBrokenRuleByKey*.

Toujours dans la classe du BO v_TVA, je vais donc coder :

```
Private Sub v_TVA_CheckRulesOnCurrentRow(ByVal e As
MicroFour.StrataFrame.Business.CheckRulesEventArgs)

    If Me.taux < 0 Then
        Me.AddBrokenRuleByKey(v_TVAFieldNames.taux, "NegatifInterdit")
    End If

    If Me.EditingState = MicroFour.StrataFrame.Business.BusinessEditingState.Editing Then
        Dim OldTaux As Decimal = Me.TauxTVAold(Me.tva_pk)

        If Me.TauxTVAestPrimaire(OldTaux) AndAlso _
        Me.taux <> OldTaux Then
            Me.AddBrokenRuleByKey(v_TVAFieldNames.taux, "TauxTVAprimaireModif")
        End If

        If Me.taux = 0 _
        AndAlso Me.TauxTVAdevientZero(Me.tva_pk) Then
            Me.AddBrokenRuleByKey(v_TVAFieldNames.taux, "TauxTVAdevientZero")
        End If

    End If 'BusinessEditingState.Editing
End Sub
```

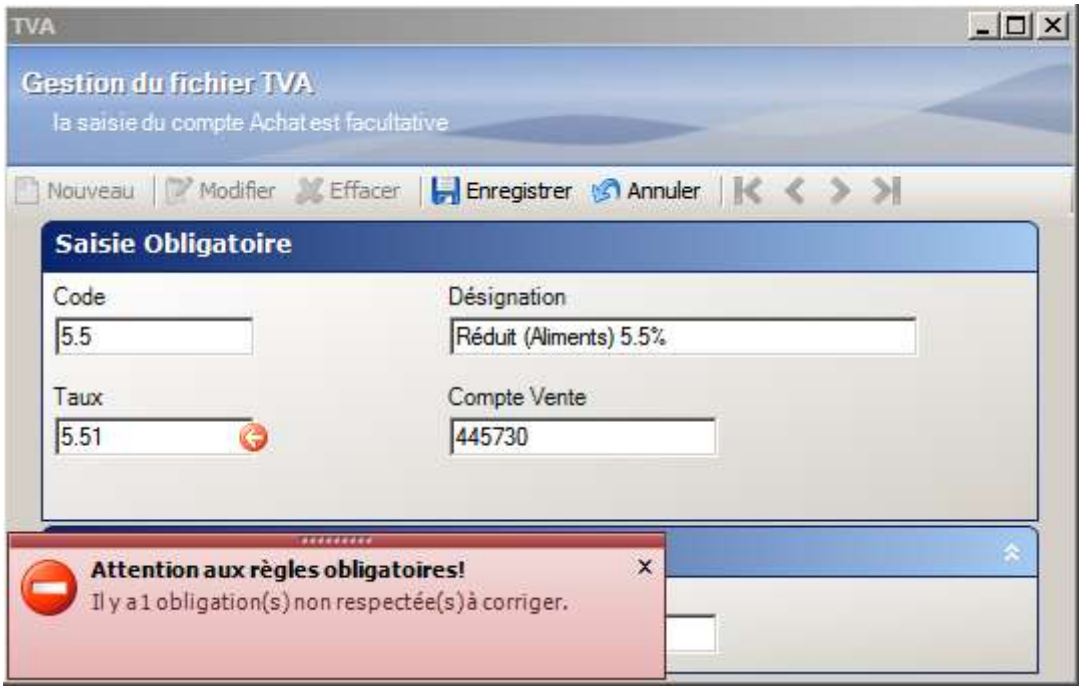
Que l'on soit en ajout ou en modification d'enregistrement, les taux négatifs sont interdits : il suffit alors d'ajouter la règle en appelant la chaine adéquate.

En modification d'enregistrement, je dois interdire les changements de taux sur les taux basiques, et également empêcher qu'on mette à zéro un taux qui ne l'était pas. Je dois donc obtenir la valeur actuelle de ce taux sur le serveur pour cet enregistrement, c'est ce que fait la fonction TauxTVAold.

```
Public Function TauxTVAold(ByVal PK As Integer) As Decimal
    Dim loCmdSQL As New SqlCommand
    loCmdSQL.CommandText = "" & _
    "SELECT  taux FROM    v_tva " & _
    "WHERE tva_pk = @tva_pk"

    loCmdSQL.Parameters.Add("@tva_pk", SqlDbType.Int)
    loCmdSQL.Parameters("@tva_pk").Value = PK

    Return Me.ExecuteScalar(loCmdSQL)

End Function
```

Et c'est tout!

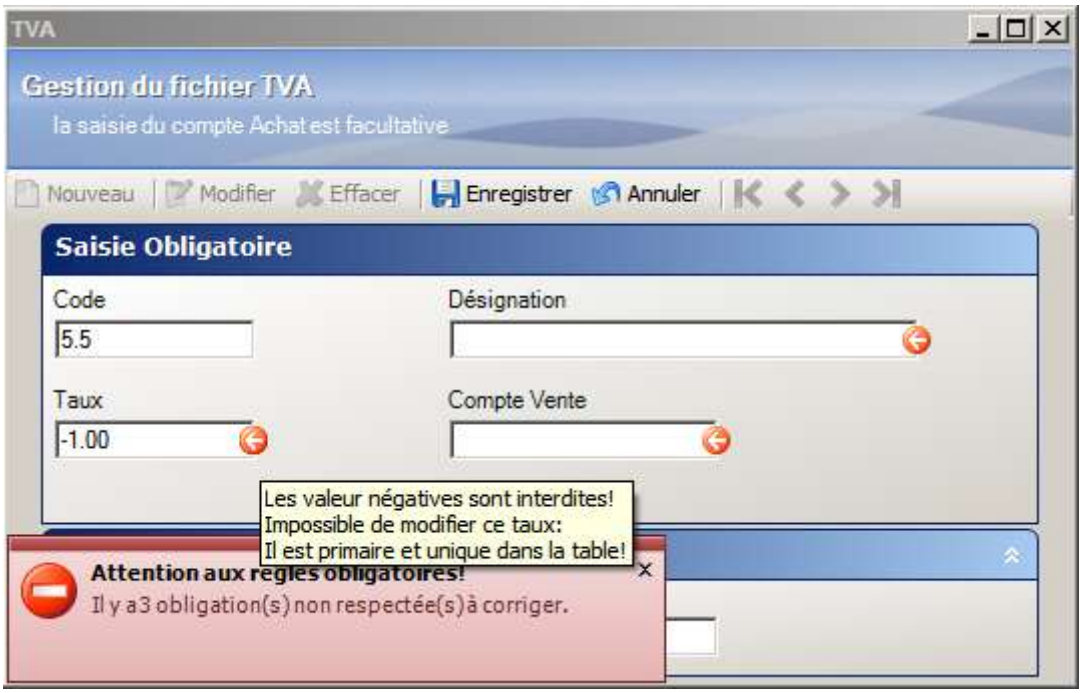
Voici un exemple de tentative de modification d'un taux basique :



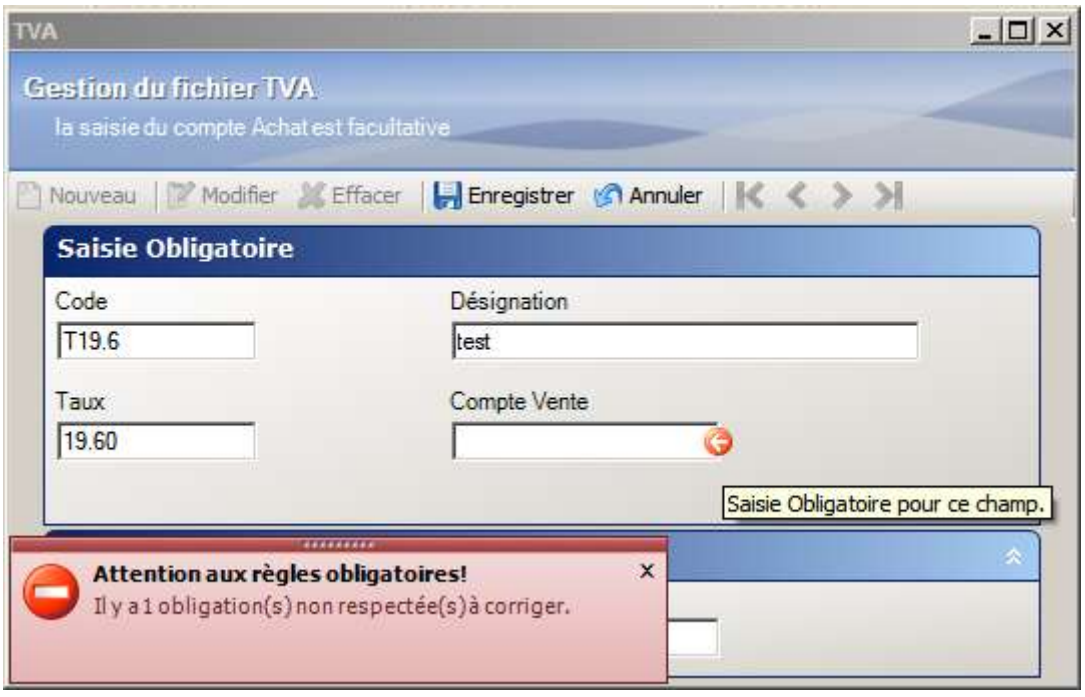
Une fois que le « PopUp » a disparu, le texte de la règle TauxTVAprimaireModif apparaît dans l'info-bulle liée à la flèche signalant le champ (ce champ a été désigné par le premier paramètre du AddBrokenRule).

Si plusieurs règles sont enfreintes, leur nombre total apparaît dans le PopUp, et chaque champ est signalé. Si un même champ cumule plusieurs infractions, l'info-bulle concatènera automatiquement les messages le concernant.

Par exemple :



Les TextBox liés aux champs designation et cpte_vente signalent une règle non respectée, parce que la saisie a été déclaré obligatoire en ajoutant les champs à la collection des *RequiredFields* au niveau du BO.



Pour ajouter ces champs à la collection des *Requiredfields*, le plus rapide est d'utiliser l'assistant depuis la feuille de propriétés du BO en mode de création (il suffit de cliquer sur le bouton `(...)` qui suit la propriété). On peut bien sur passer par du code si on le souhaite.

Sur un ajout d'enregistrement

À chaque création d'enregistrement, l'évènement *SetDefaultValue* est déclenché, et nous pouvons donc renseigner directement dans le BO ces valeurs par défaut pour les champs souhaités ; ces valeurs seront automatiquement remontées vers l'interface visuelle dans les contrôles liés, sans aucun code supplémentaire.

```
    Private Sub v_TVA_SetDefaultValues()

        Me.cpte_vente = Me.getDfltVte()
        Me.cpte_achat = Me.getDfltAch()

    End Sub
```

Dans cette application, les champs cpte_vente et cpte_achat de la table TVA n'ont pas de valeur par défaut définie sur le serveur en tant que CONSTRAINT ou DEFAULT, mais pour chacun d'eux je veux récupérer une valeur à proposer qui se trouve dans une table de paramètres généraux (bien évidemment, cette table n'est pas directement accessible au contexte de sécurité du schéma du login de l'application sur SQL Server, mais seulement par une vue en lecture seule).

La fonction getDfltVte est simplement :

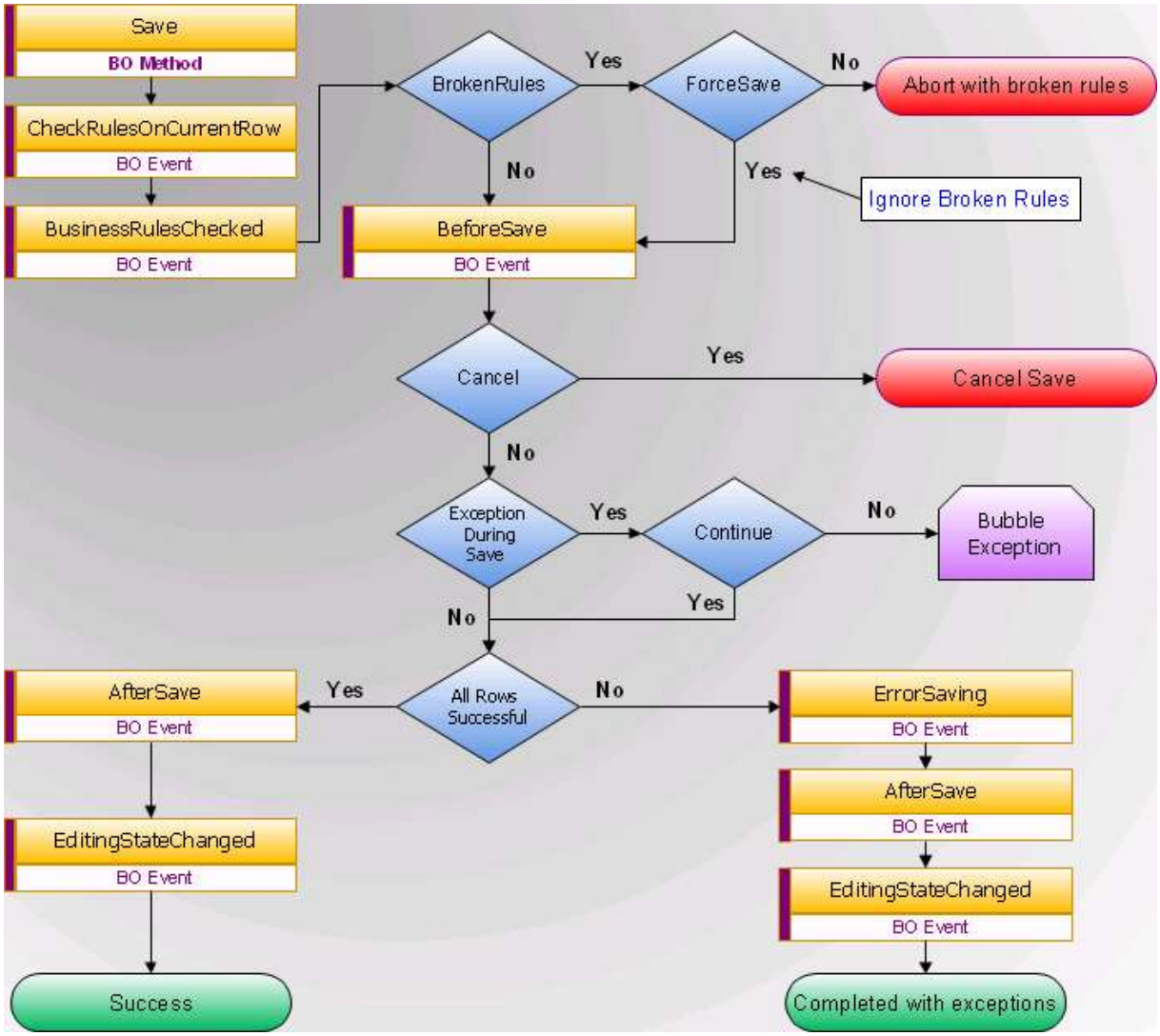
```
Public Function getDfltVte() As String

    Dim loCmdSQL As New SqlCommand
    loCmdSQL.CommandText = "" & _
    "SELECT cpttvavte as  VteDflt  " & _
    "FROM v_ParamGene"

    Return Me.ExecuteScalar(loCmdSQL)
End Function
```

Pour conclure…

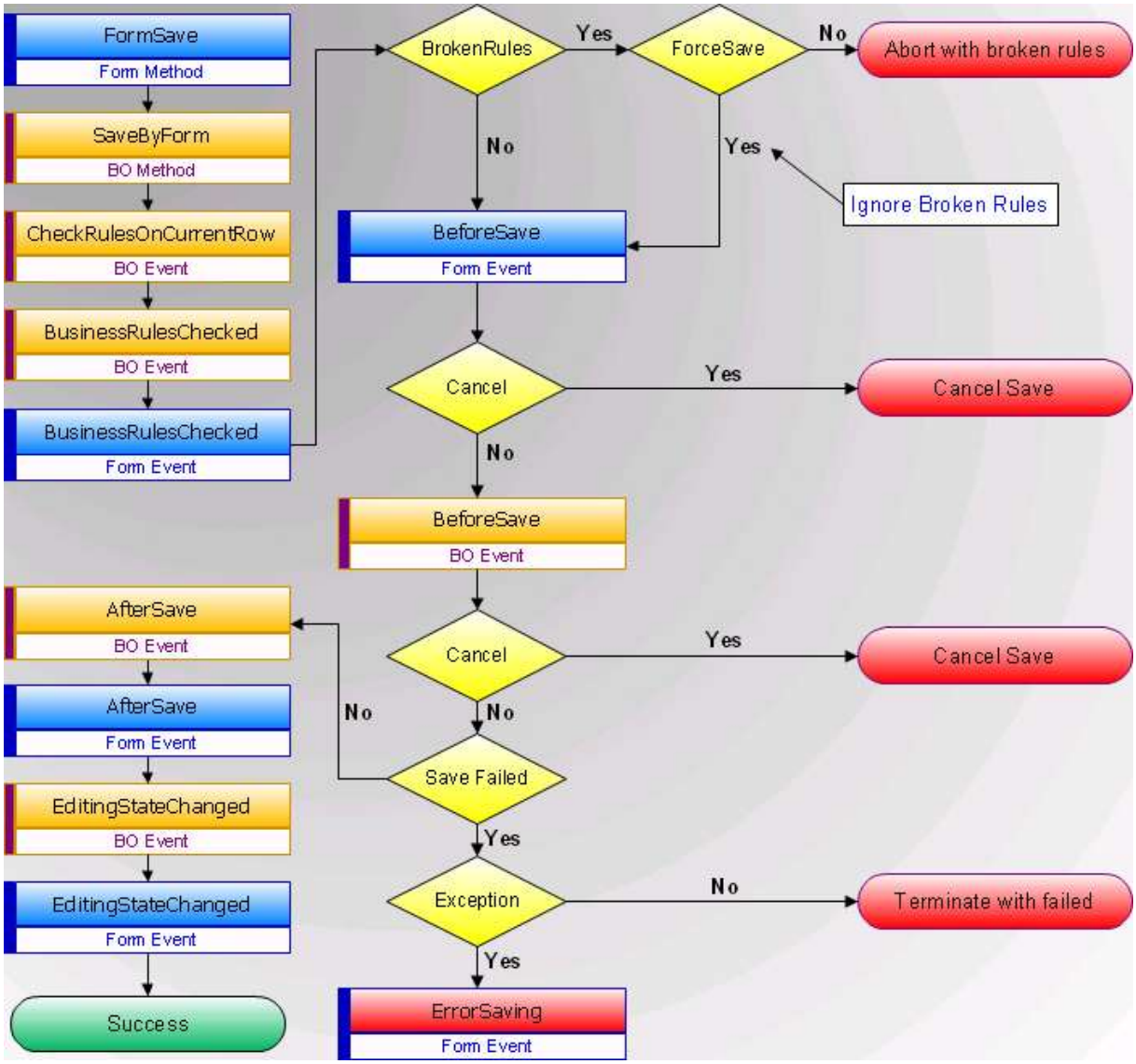
Faites un simple test sur une des fonctionnalités présentées dans cette étude : essayez de réaliser la même chose sans StrataFrame, avec seulement les outils de .Net ! comptez le temps passé, comptez vos lignes de code…

Annexes

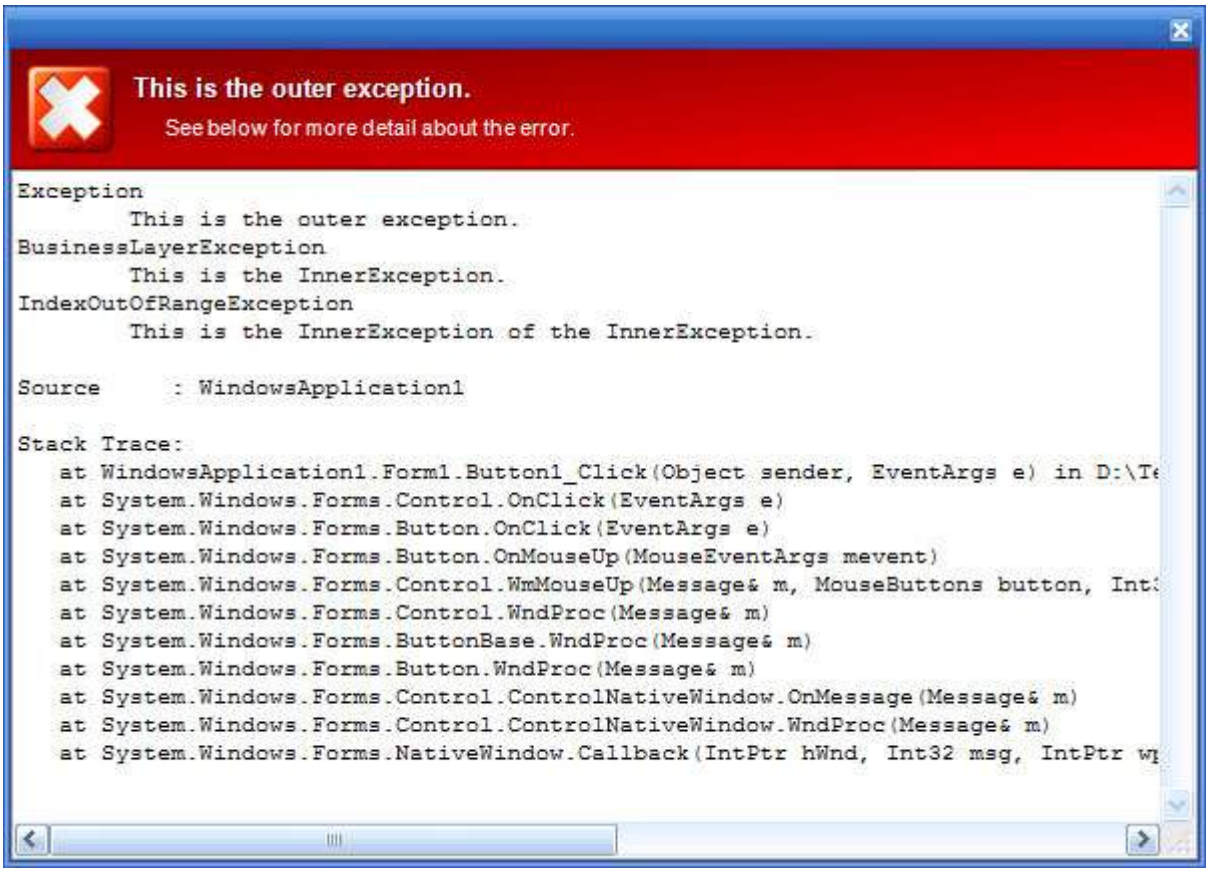
Save depuis le BO



Save depuis le Form



ApplicationErrorWindow



Logicaveau Login