

**FRAMEWORK**

**Projet SYSGD II**

**Version 1.0**

**Comité paritaire des agents de sécurité**

**Août 2011   
Jean Sylvain**

# HISTORIQUE DE REVISION

|  |
| --- |
| Nom du fichier |
| Framework.docx |

|  |  |  |  |
| --- | --- | --- | --- |
| Ver | Date | Réviseur (s) | Description |
| 1.0 | 2011-08-25 | JS | Rédaction initiale |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# APPROBATION ET VALIDATION DE CE DOCUMENT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date Approbation | Ver | Organisation | Nom | Description | Signature |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# DOCUMENTS DE RÉFÉRENCE

|  |  |  |  |
| --- | --- | --- | --- |
| Titre | Date | Version | Auteur (s) |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table des matières

[HISTORIQUE DE REVISION 1](#_Toc302057064)

[APPROBATION ET VALIDATION DE CE DOCUMENT 1](#_Toc302057065)

[DOCUMENTS DE RÉFÉRENCE 1](#_Toc302057066)

[But du document 3](#_Toc302057067)

[Introduction 3](#_Toc302057068)

[DataObjects : CLASSE CoginovEntityObject 3](#_Toc302057069)

[DataObjects : Implantation 4](#_Toc302057070)

[DataControler: BaseDataCtrl.cs 5](#_Toc302057071)

[DataControler : Implantation 5](#_Toc302057072)

[Annexe 1 : Source de CoginovEntityObject 7](#_Toc302057073)

[Annexe 2 : Source de BaseDataCtrl.cs 11](#_Toc302057074)

# But du document

Ce document présente le cadre de développement (framework) utilisé pour le projet SYSGD II.

## Introduction

Le framework est basé sur le produit Entity Framework de Microsoft. Il vise à permettre de créer des objets d’affaires avec persistance à la BD, Le framework regroupera également différentes fonctions utilitaires propre au projet. Le framework utilise le modèle du DataControler/DataObject.

## DataObjects : CLASSE CoginovEntityObject

Le framework utilise les objets entitaires (entity objects) produits par EF. Ces objets sont cependant modifiés pour hériter de la *CoginovEntityObject.cs.* Pour ce faire on utilise le fichier *CoginovModel.tt* comme template de génération des objets EF*.* La classe *CoginovEntityObject* hérite de la classe *EntityObject.* Les objets entitaire de notre framework héritent donc de toutes les propriétés et méthodes des objets entitaires génériques.

La classe *CoginovEntityObject.cs* ajoute cependant un comportement de validation des contentions à la bd ainsi que des propriétés permettant de contrôler le comportement des objets. Le code-source est listé à l’annexe 1 du présent document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | SCOPE | TYPE | Valeur retournée | DESCRIPTION |
| ValErr | Protected | Property | Hashtable | Dictionnary : Liste des erreurs de validation. Les erreurs sont conservées avec une clé identifiant le contrôle. |
| ClearValErr() | Protected | Method | Void | Réinitialise le dictionnaire des erreurs ValErr. |
| AddValErr(string ctrlKey, string Message) | Protected | Method | Void | Ajoute un message d’erreur pour un contrôle donné. |
| GetErrorMessage(string ctrlKey) | Public | Method | String | Retourne le message d’erreur correspondant à un contrôle donné. |
| \_OriginalCheckSumValue | Protected | Property | String | Checksum de l’objet lors de l’instanciation. |
| \_OldCheckSumValue | Protected | Property | String | Checksum de l’objet avant sa réécriture à la bd. |
| \_CanDelete | Protected | Property | Bool | Indique si l’on peut supprimer un objet de ce type. Valeur défault : False. |
| SetOriginalCheckSumValue(string val) | Public | Method | void | Initialise \_OriginalCheckSumValue |
| SetOldCheckSumValue(string val) | Public | Method | void | Initialise \_OldCheckSumValue |
| GetOriginalCheckSumValue() | Public | Method | String | Expose le checksum initial de l'objet |
| Save() | Public | Method | Bool | Applique le test de concurrence et la validation et retourne true si les tests sont concluants. Il est de la responsabilité du contrôleur d’appliquer la sauvegarde par la suite. |
| Delete() | Public | Method | Bool | Retourne la valeur de \_CanDelete. |

La classe *CoginovEntityObject.cs* appartient au projet SYSGD\_DAL et réside dans le répertoire Framework.

## DataObjects : Implantation

Les objets d’affaires (OA) issus du modèle seront implantés au sein du projet SYSGD\_DAL dans le répertoire BusinessObjects. Chaque objet sera contenu dans un fichier qui lui est propre. On utilise la notion de classe partielle pour étendre les fonctionnalités des objets entitaires créés par EF. Les OA doivent, au minimum, implanter un constructeur et une méthode de validation.

Le code suivant illustre l’implantation d’un objet représentant les employeurs :

namespace SYSGD\_DAL

{

    public partial class Employeur

    {

        /// <summary>

        /// Constructeur utilisé pour définir les

        /// parametres et le comportement général de l'objet.

        /// </summary>

        public Employeur()

        {

            this.\_CanDelete = false;

        }

        /// <summary>

        /// Ajouter ici le code pour l'implantation des regles de validation

        /// et les regles d'affaire.

        /// </summary>

        protected override bool Validate()

        {

            // Mettre ici le code de validation de l'objet.

//---------------------------------------------------------

            return (this.ValErr.Count == 0);

        }

    }

}

Le fichier CoginovObject.zip contient un template VS automatisant la création d’items de ce type. Il doit etre déposé dans le répertoire \Documents and Settings\<USERNAME>\Documents\Visual Studio 2010\Templates\ItemTemplates\Visual C#\Coginov

## DataControler: BaseDataCtrl.cs

La classe abstraite BaseDataCtrl.cs reside dans le projet SYSGD\_DAL dans le répertoire Framework. Cette classe permet de créer des objets controleurs. Ces objets ont la responsabilité de gérer un type d’objet d’affaire tout au long de son cycle de vie. A cet effet la classe BaseDataCtrl.cs force les classes qui l’implantent à coder les fonctions de selection, création, mise à jour et suppression.

Le code-source est listé à l’annexe 2 du présent document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NAME | SCOPE | TYPE | Valeur retournée | DESCRIPTION |
| Context | Protected | Property | SYSDGIIENtities | Crée et/ou retourne un objet de contexte permettant l’acces à la bd. Ajoute un Event Handler « Context\_SavingChanges ». |
| ErrorMessage | Public | Property | String | Message d’erreur généré lors d’une opération. |
| Select() | Public Abstract | Method | List<T> | Retourne une liste des objets de classe T |
| Select(int ID) | Public Abstract | Method | <T> | Retourne un objet de classe T |
| Create() | Public Abstract | Method | <T> | Crée un objet de classe T |
| Save() | Public | Method | Bool | Invoque le save de l’objet d’affaire et si ce dernier retourne true, il invoque this.save(). |
| save() | Protected | Method | Bool | Persiste l’OA à la BD. Si erreur, le message est capté et inscrit dans ErrorMessage. |
| Delete(CoginovEntityObject ef) | Public | Method | Bool | Invoque le delete de l’objet d’affaire et si ce dernier retourne true, il invoque this.delete(). |
| delete | Protected | Method | Bool | Persiste la suppression de l’OA à la BD. Si erreur, le message est capté et inscrit dans ErrorMessage. |
| GetChecksum(EntityObject eo) | Public | Method | String | Retourne le checksum de l’objet passé en argument. |

## DataControler : Implantation

Les objets DataControler sont créés et conservés au sein du projet SYSDG\_DAL dans le répertoire DataControllers. Chaque OA doit avoir son propre contrôleur. Chaque contrôleur est contenu dans un fichier cs qui lui est propre. Le nom de ce fichier est construit comme suit : <NomObjetAffaire>CRTL.cs

Le code suivant illustre l’implantation d’un contrôleur :

namespace SYSGD\_DAL

{

    public class EmployeurCTRL : BaseDataCtrl<Employeur>

    {

       #region override abstract methods

            public override List<Employeur> Select()

            {

                var myList =  context.EmployeurSet.Include("FicheEmployeurs").ToList();

                foreach ( Employeur ef in myList)

                {

                    //ef.context = this.context;

                    ef.SetOriginalCheckSumValue( this.GetChecksum(ef));

                }

                return myList;

            }

            public override Employeur Select(int id)

            {

                var myEF = context.EmployeurSet.Where(a => a.EmployeurID.Equals(id)).FirstOrDefault();

               // myEF.context = this.context;

                myEF.SetOriginalCheckSumValue( this.GetChecksum(myEF));

                return myEF;

            }

            public override Employeur Create()

            {

                var myEF = new Employeur();

               // myEF.context = this.context;

                // Ajouter ici les valeurs par defaut

                myEF.SetOriginalCheckSumValue(this.GetChecksum(myEF));

                return myEF;

            }

        #endregion

   }

}

# Annexe 1 : Source de CoginovEntityObject

//

// Coginov® - http://www.coginov.com

// Copyright (c) 2011

// par Coginov inc

//

// Ce code est la propriété exclusive de Coginov inc.

//

// Classe abstraite servant de base pour les classes de la DAL

// représentant des objets de type DataController.

//----------------------------------------------------------------------------------------

using System;

using System.Collections.Generic;

using System.Data.Objects;

using System.Data.Objects.DataClasses;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Security.Cryptography;

namespace SYSGD\_DAL.Framework

{

    public abstract class BaseDataCtrl<T>

    {

        #region Private/protected members

           private SYSGDIIEntities \_context;

           protected SYSGDIIEntities  context

           {

                get

                {

                    if (\_context == null)

                    {

                        \_context = new SYSGDIIEntities();

                        \_context.SavingChanges += new EventHandler(context\_SavingChanges);

                    }

                    return \_context;

                }

           }

        #endregion

        #region Public members

            public string errorMessage { get; set; }

        #endregion

        #region Abstract Methods

            /// <summary>

            /// Retourne une liste d'objets de type T

            /// </summary>

            /// <returns></returns>

            public abstract List<T> Select();

            /// <summary>

            /// Retourne un objet de type T correspondant

            /// à la clé qui lui est passée.

            /// </summary>

            /// <param name="id">Cle primaire de l'objet a retourner</param>

            /// <returns></returns>

            public abstract T Select(int id);

            /// <summary>

            /// Retourne un objet vide initialisé avec les valeurs par defaut.

            /// </summary>

            /// <returns></returns>

            public abstract T Create();

        #endregion

        #region Public Methods

            /// <summary>

            /// Coordonne la sauvegarde de l'objet entitaire (BO) qui lui

            /// est passé en argument.

            /// </summary>

            /// <param name="ef">Objet entitaire (BO)</param>

            /// <returns>lRéussite</returns>

            public bool Save(CoginovEntityObject ef)

            {

                if (ef.Save())

                   return this.save();

                else

                    return false;

           }

            /// <summary>

            /// Coordonnateur de la suppression

            /// </summary>

            /// <param name="ef"></param>

            /// <returns></returns>

            public bool Delete(CoginovEntityObject ef)

            {

                if (ef.Delete())

                   return this.delete(ef);

                else

                    return false;

            }

            /// <summary>

            /// Calcule le checksum de l'objet entitaire data. Ce string

            /// permet de determiner si deux objets sont identiques.

            /// </summary>

            /// <returns>Checksum</returns>

            public string GetChecksum(EntityObject eo)

            {

                if (eo == null)

                    return "";

                else

                {

                BinaryFormatter bf = new BinaryFormatter();

                MemoryStream ms = new MemoryStream();

                bf.Serialize(ms, eo);

                SHA256Managed sha = new SHA256Managed();

                byte[] checksum = sha.ComputeHash(ms.ToArray());

                return BitConverter.ToString(checksum).Replace("-", String.Empty);

                }

            }

        #endregion

        #region Private methods

            /// <summary>

            /// Evenement appelé lors du

            /// </summary>

            /// <param name="sender"></param>

            /// <param name="e"></param>

            private void context\_SavingChanges(object sender, EventArgs e)

            {

                ObjectContext context = sender as ObjectContext;

                if (context != null)

                {

                    foreach (ObjectStateEntry entry in context.ObjectStateManager.GetObjectStateEntries(System.Data.EntityState.Modified))

                    {

                        // Implante le mode de sauvegarde transactionnel sur les objets contenant la string .Fiche....

                        if (entry.Entity.GetType().ToString().ToUpper().Contains(".FICHE"))

                        {

                            context.ObjectStateManager.ChangeObjectState(entry.Entity, System.Data.EntityState.Added);

                        }

                    }

                }

            }

            /// <summary>

            /// Primitive chargé de la sauvegarde.

            /// </summary>

            /// <returns>lRéussite</returns>

            protected bool save()

            {

                try

                {

                    context.SaveChanges();

                    return true;

                }

                catch (Exception exc)

                {

                    this.errorMessage = exc.Message;

                    //TODO: Ajouter un message au journal d'evenement.

                    return false;

                }

            }

            /// <summary>

            /// Primitive chargé de la suppression

            /// </summary>

            /// <param name="ef">Objet a supprimer</param>

            /// <returns>lRéussite</returns>

            protected bool delete(CoginovEntityObject ef)

            {

                try

                {

                    context.DeleteObject(ef);

                    context.SaveChanges();

                    return true;

                }

                catch (Exception exc)

                {

                    this.errorMessage = exc.Message;

                    //TODO: Ajouter un message au journal d'evenement.

                    return false;

                }

            }

        #endregion

        #region Common Functions

            /// <summary>

            /// Service servant à determiner la préexistance d'un

            /// objet dans la bd.

            /// </summary>

            /// <param name="tableName">Nom de la table</param>

            /// <param name="fieldName">Nom de la colonne</param>

            /// <param name="fieldNameId"></param>

            /// <param name="value"></param>

            /// <param name="id"></param>

            /// <returns></returns>

            protected bool IsDuplicate(string tableName, string fieldName,

                string fieldNameId, string value, int id)

            {

                string sql =

                    "SELECT COUNT(" + fieldNameId + ") AS DuplicateCount " +

                      "FROM " + tableName +

                    " WHERE " + fieldName + " = {0}" +

                      " AND " + fieldNameId + " <> {1}";

                var result = context.ExecuteStoreQuery<DuplicateCheck>(sql, new object[] { value, id });

                List<DuplicateCheck> list = result.ToList();

                return (list[0].DuplicateCount > 0);

            }

            protected bool IsDuplicate( string tableName, string fieldName,

                string fieldNameId, DateTime value, int id)

            {

                string sql =

                    "SELECT COUNT(" + fieldNameId + ") AS DuplicateCount " +

                      "FROM " + tableName +

                    " WHERE " + fieldName + " = {0}" +

                      " AND " + fieldNameId + " <> {1}";

                var result = context.ExecuteStoreQuery<DuplicateCheck>(sql, new object[] { value, id });

                List<DuplicateCheck> list = result.ToList();

                return (list[0].DuplicateCount > 0);

            }

        #endregion Common Functions

    }

}

# Annexe 2 : Source de BaseDataCtrl.cs

using System;

using System.Collections.Generic;

using System.Data.Objects;

using System.Data.Objects.DataClasses;

using System.IO;

using System.Linq;

using System.Runtime.Serialization.Formatters.Binary;

using System.Security.Cryptography;

namespace SYSGD\_DAL.Framework

{

    public abstract class BaseDataCtrl<T>

    {

        #region Private/protected members

           private SYSGDIIEntities \_context;

           protected SYSGDIIEntities  context

           {

                get

                {

                    if (\_context == null)

                    {

                        \_context = new SYSGDIIEntities();

                        \_context.SavingChanges += new EventHandler(context\_SavingChanges);

                    }

                    return \_context;

                }

           }

        #endregion

        #region Public members

            public string errorMessage { get; set; }

        #endregion

        #region Abstract Methods

            /// <summary>

            /// Retourne une liste d'objets de type T

            /// </summary>

            /// <returns></returns>

            public abstract List<T> Select();

            /// <summary>

            /// Retourne un objet de type T correspondant

            /// à la clé qui lui est passée.

            /// </summary>

            /// <param name="id">Cle primaire de l'objet a retourner</param>

            /// <returns></returns>

            public abstract T Select(int id);

            /// <summary>

            /// Retourne un objet vide initialisé avec les valeurs par defaut.

            /// </summary>

            /// <returns></returns>

            public abstract T Create();

        #endregion

        #region Public Methods

            /// <summary>

            /// Coordonne la sauvegarde de l'objet entitaire (BO) qui lui

            /// est passé en argument.

            /// </summary>

            /// <param name="ef">Objet entitaire (BO)</param>

            /// <returns>lRéussite</returns>

            public bool Save(CoginovEntityObject ef)

            {

                if (ef.Save())

                   return this.save();

                else

                    return false;

           }

            /// <summary>

            /// Coordonnateur de la suppression

            /// </summary>

            /// <param name="ef"></param>

            /// <returns></returns>

            public bool Delete(CoginovEntityObject ef)

            {

                if (ef.Delete())

                   return this.delete(ef);

                else

                    return false;

            }

            /// <summary>

            /// Calcule le checksum de l'objet entitaire data. Ce string

            /// permet de determiner si deux objets sont identiques.

            /// </summary>

            /// <returns>Checksum</returns>

            public string GetChecksum(EntityObject eo)

            {

                if (eo == null)

                    return "";

                else

                {

                BinaryFormatter bf = new BinaryFormatter();

                MemoryStream ms = new MemoryStream();

                bf.Serialize(ms, eo);

                SHA256Managed sha = new SHA256Managed();

                byte[] checksum = sha.ComputeHash(ms.ToArray());

                return BitConverter.ToString(checksum).Replace("-", String.Empty);

                }

            }

        #endregion

        #region Private methods

            /// <summary>

            /// Evenement appelé lors de la sauvegarde à la bd.

            /// </summary>

            /// <param name="sender"></param>

            /// <param name="e"></param>

            private void context\_SavingChanges(object sender, EventArgs e)

            {

                ObjectContext context = sender as ObjectContext;

                if (context != null)

                {

                    foreach (ObjectStateEntry entry in context.ObjectStateManager.GetObjectStateEntries(System.Data.EntityState.Modified))

                    {

                        // Implante le mode de sauvegarde transactionnel sur les objets contenant la string .Fiche....

                        if (entry.Entity.GetType().ToString().ToUpper().Contains(".FICHE"))

                        {

                            context.ObjectStateManager.ChangeObjectState(entry.Entity, System.Data.EntityState.Added);

                        }

                    }

                }

            }

            /// <summary>

            /// Primitive chargé de la sauvegarde.

            /// </summary>

            /// <returns>lRéussite</returns>

            protected bool save()

            {

                try

                {

                    context.SaveChanges();

                    return true;

                }

                catch (Exception exc)

                {

                    this.errorMessage = exc.Message;

                    //TODO: Ajouter un message au journal d'evenement.

                    return false;

                }

            }

            /// <summary>

            /// Primitive chargé de la suppression

            /// </summary>

            /// <param name="ef">Objet a supprimer</param>

            /// <returns>lRéussite</returns>

            protected bool delete(CoginovEntityObject ef)

            {

                try

                {

                    context.DeleteObject(ef);

                    context.SaveChanges();

                    return true;

                }

                catch (Exception exc)

                {

                    this.errorMessage = exc.Message;

                    //TODO: Ajouter un message au journal d'evenement.

                    return false;

                }

            }

        #endregion

        #region Common Functions

            /// <summary>

            /// Service servant à determiner la préexistance d'un

            /// objet dans la bd.

            /// </summary>

            /// <param name="tableName">Nom de la table</param>

            /// <param name="fieldName">Nom de la colonne</param>

            /// <param name="fieldNameId"></param>

            /// <param name="value"></param>

            /// <param name="id"></param>

            /// <returns></returns>

            protected bool IsDuplicate(string tableName, string fieldName,

                string fieldNameId, string value, int id)

            {

                string sql =

                    "SELECT COUNT(" + fieldNameId + ") AS DuplicateCount " +

                      "FROM " + tableName +

                    " WHERE " + fieldName + " = {0}" +

                      " AND " + fieldNameId + " <> {1}";

                var result = context.ExecuteStoreQuery<DuplicateCheck>(sql, new object[] { value, id });

                List<DuplicateCheck> list = result.ToList();

                return (list[0].DuplicateCount > 0);

            }

            protected bool IsDuplicate( string tableName, string fieldName,

                string fieldNameId, DateTime value, int id)

            {

                string sql =

                    "SELECT COUNT(" + fieldNameId + ") AS DuplicateCount " +

                      "FROM " + tableName +

                    " WHERE " + fieldName + " = {0}" +

                      " AND " + fieldNameId + " <> {1}";

                var result = context.ExecuteStoreQuery<DuplicateCheck>(sql, new object[] { value, id });

                List<DuplicateCheck> list = result.ToList();

                return (list[0].DuplicateCount > 0);

            }

        #endregion Common Functions

    }

}