

Projet API Simple

Version bêta - 12/2015



Table des matières

Objectif.....	1
Présentation du contexte	2
Etape 1 : Création du dossier API et génération des interfaces.	3
Etape 2 : Ouverture du projet d'extension	9
Etape 3 : Ajouter l'extension à un dossier de l'application EBP Gestion Commerciale Open Line	18

Objectif

L'objectif de ce tutoriel est d'expliquer comment personnaliser l'application EBP Gestion Commerciale Open Line en fonction d'une demande client.

Pour cela, on utilisera les tables personnalisées, une fonctionnalité déjà présente dans l'application.

Après le paramétrage de ces tables, on générera des DLL qui pourront être intégrées dans Visual Studio, pour permettre de développer les fonctionnalités souhaitées.

Pour ce faire, nous allons utiliser un exemple concret et suivre les différentes étapes nécessaires à la génération de l'extension.

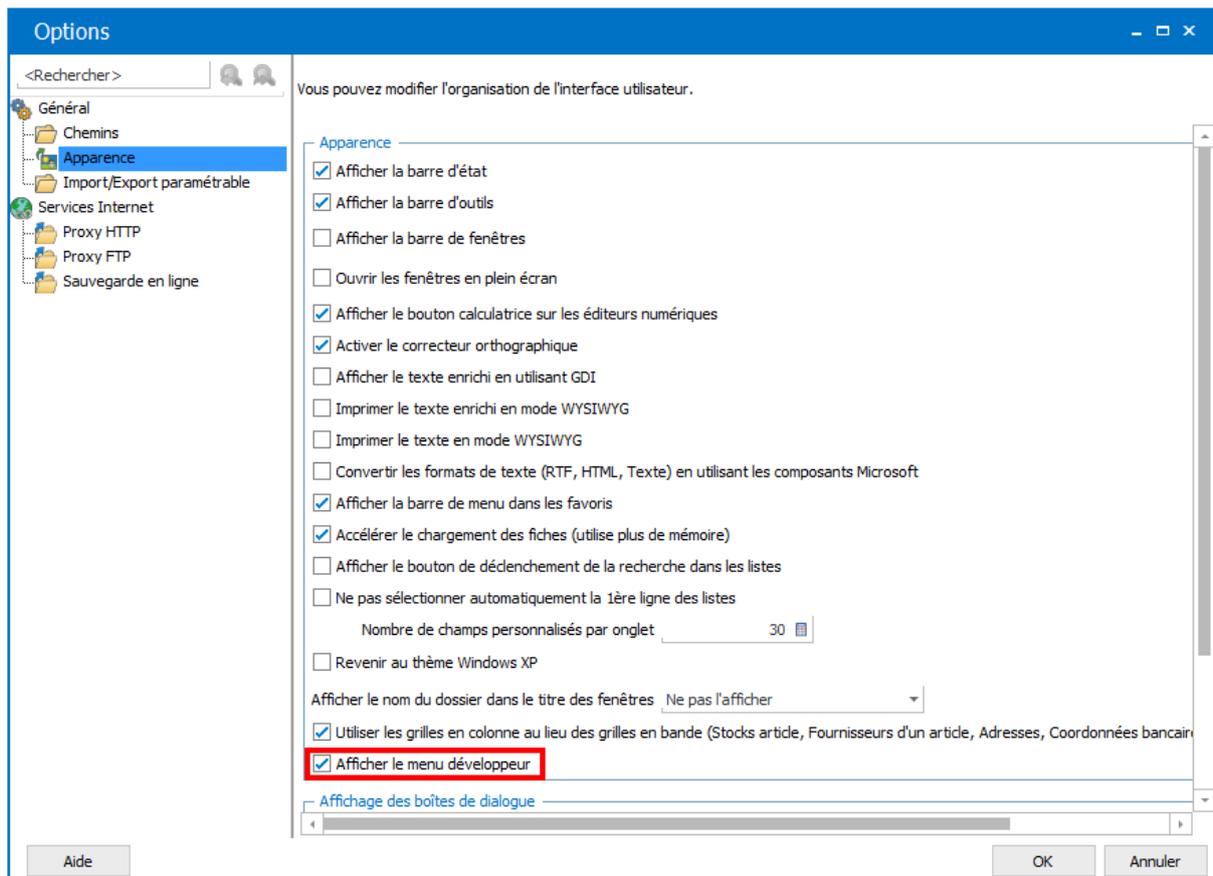
Présentation du contexte

L'objectif est d'ajouter des fonctionnalités supplémentaires autour des contacts de vente. Pour cela, nous allons ajouter 3 tables :

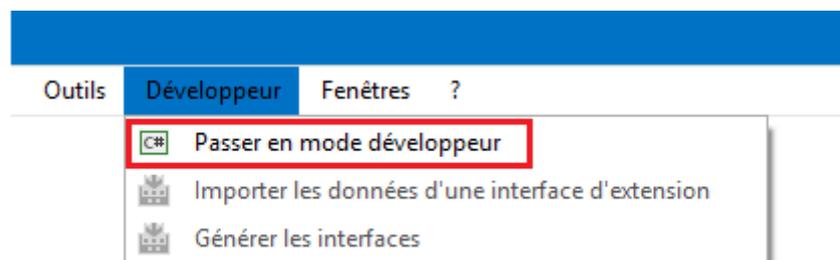
- « Fonction Contact » qui permet de définir le poste du contact (Cadre, Employé, Agent de Maitrise, ...),
- « Sous-fonction Contact » qui permet de définir la fonction du contact (Commercial, Développeur,),
- « Service Contact » qui correspond au service auquel le contact est rattaché (Développement, Commercial, Marketing, ...)

Etape 1 : Création du dossier API et génération des interfaces.

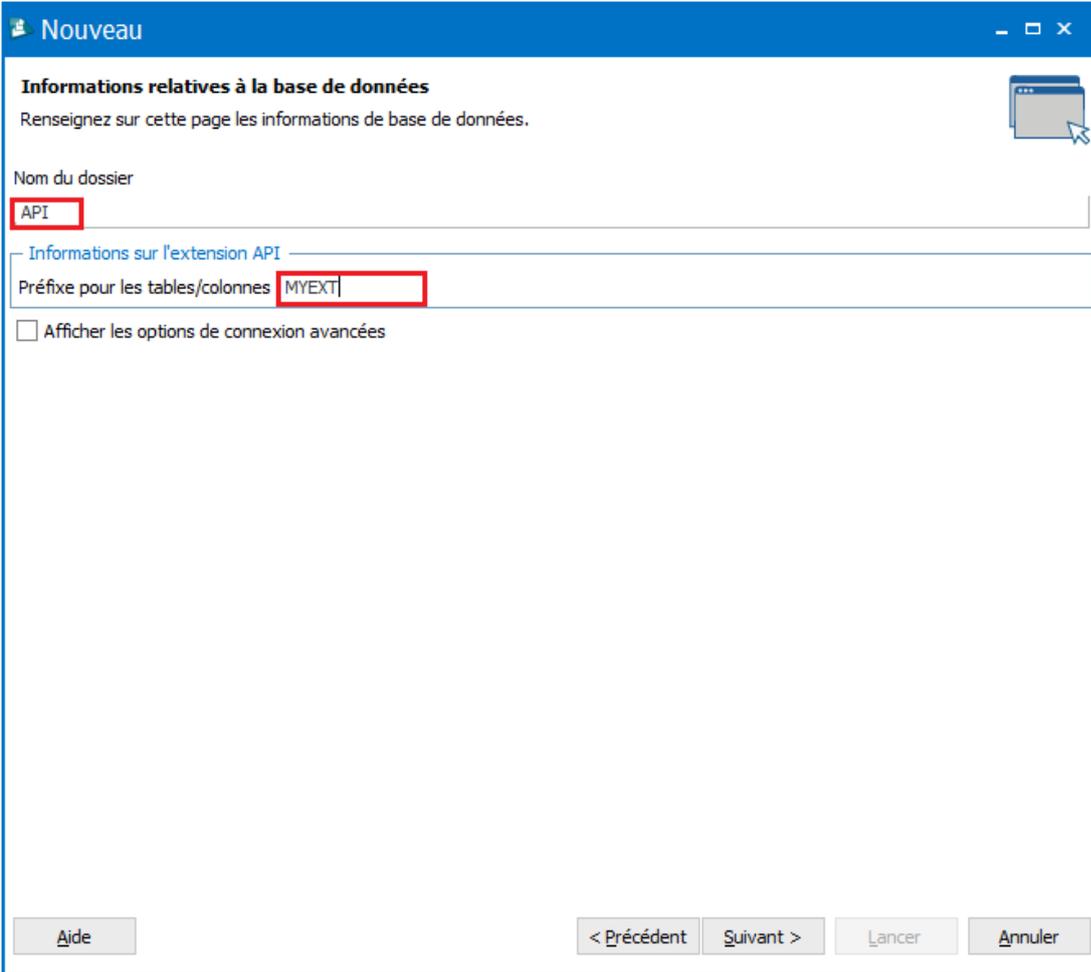
1. Lancer l'application **EBP Gestion Commerciale Open Line**, sans ouvrir de dossier.
2. Afficher le menu développeur : dans *Outils / Options / Apparence*, cocher la case **Afficher le menu développeur**.



3. Passer en mode développeur : dans *Développeur*, sélectionner **Passer en mode développeur**.



4. Créer un dossier nommé *API* avec comme « Préfixe pour les tables/colonnes » *MYEXT*. Il est important de donner un préfixe non générique afin d'éviter d'éventuels conflits avec d'autres extensions (Exemple : les 3 premières lettres pouvant correspondre à votre société).



Nouveau

Informations relatives à la base de données
Renseignez sur cette page les informations de base de données.

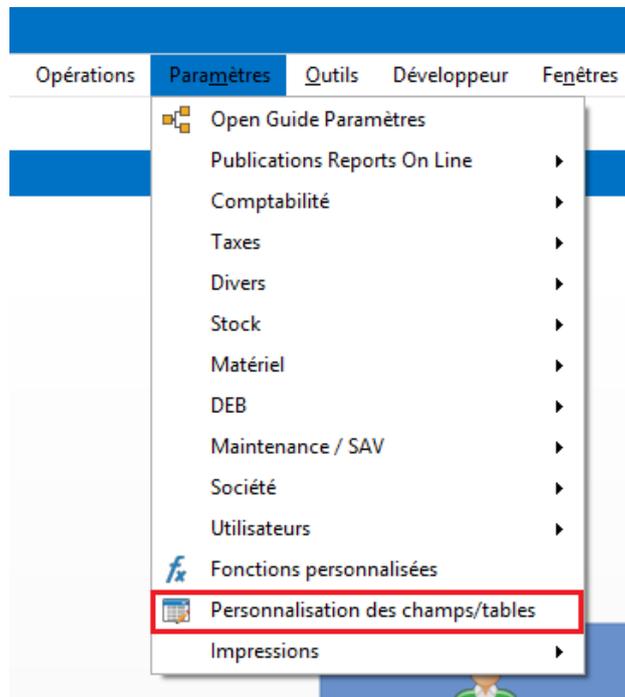
Nom du dossier
API

— Informations sur l'extension API —
Préfixe pour les tables/colonnes MYEXT

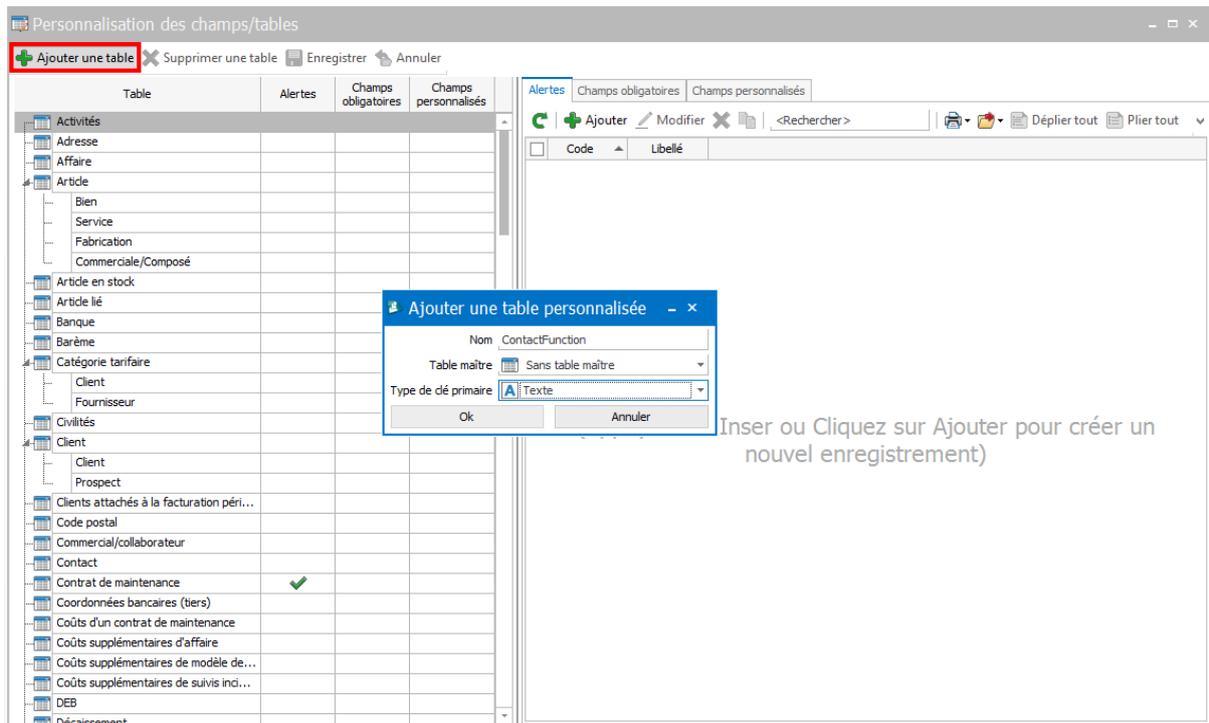
Afficher les options de connexion avancées

Aide < Précédent Suivant > Lancer Annuler

5. Aller dans le menu *Paramètres / Personnalisation des champs/tables*

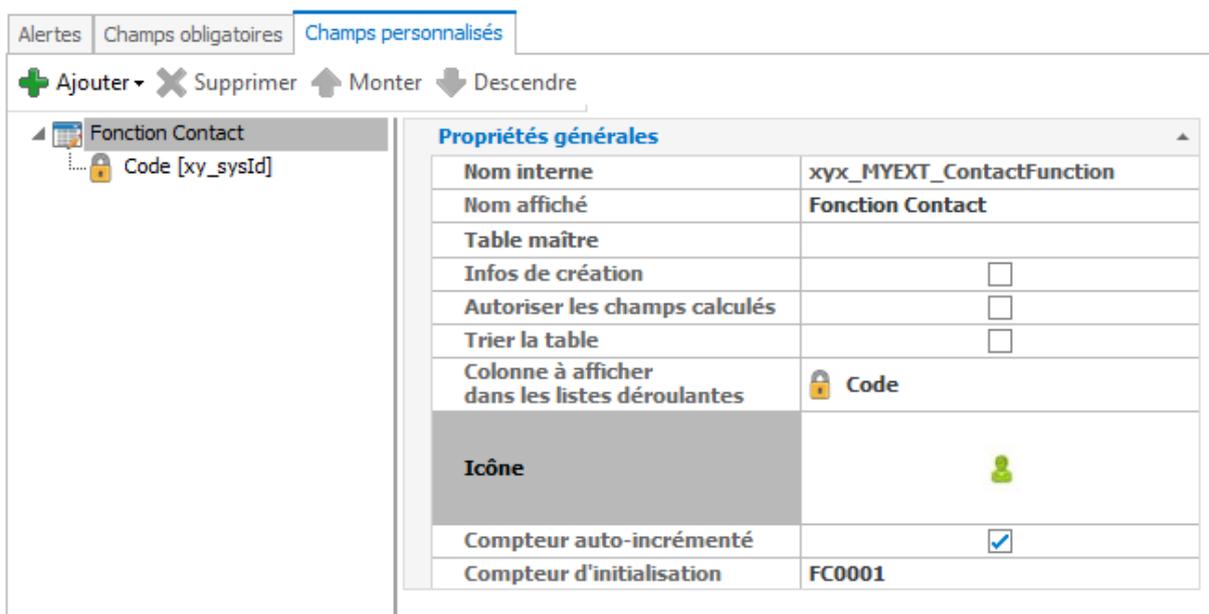


6. Pour ajouter une table personnalisée nommée *ContactFunction*, cliquer sur le bouton *Ajouter une table*.

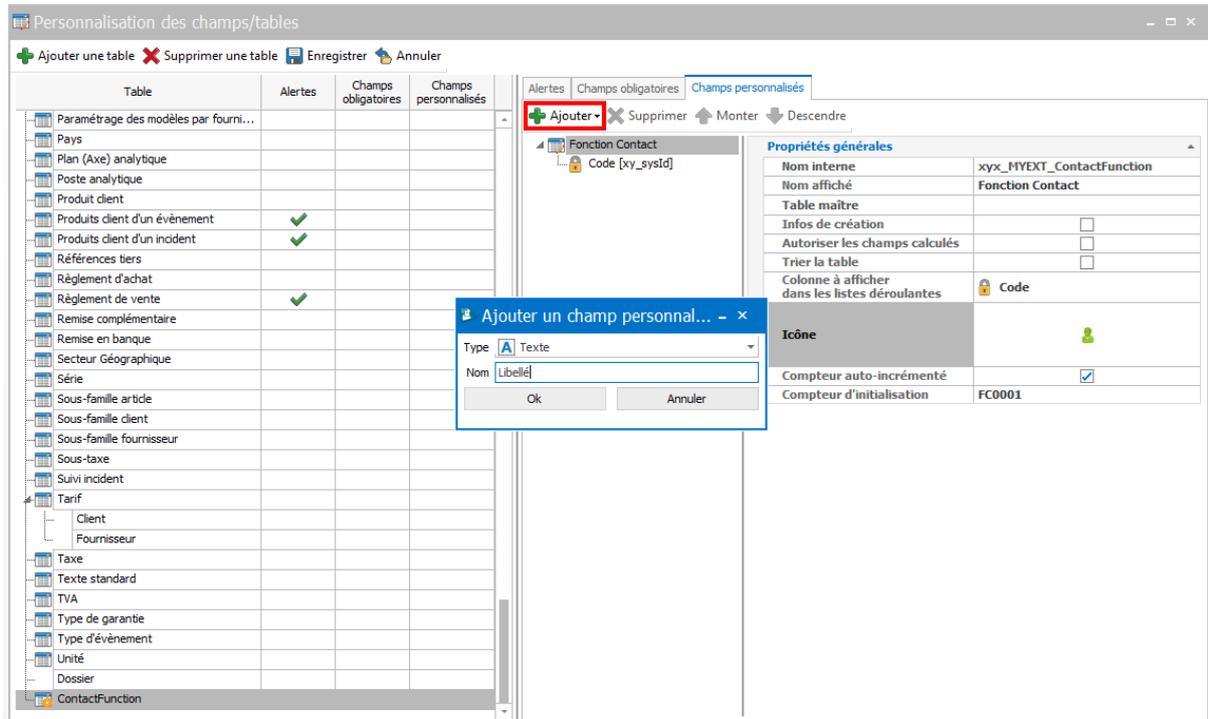


⚠ Avertissement : Les libellés des tables doivent être distincts. Il est donc fortement recommandé de les rendre suffisamment explicites pour éviter que deux tables d'extensions différentes possèdent le même libellé.

7. Renseigner les Propriétés générales de la table.



- Pour créer les champs que vous souhaitez renseigner dans votre table, cliquer sur le bouton *Ajouter* puis indiquer les différentes propriétés qui lui sont propres.

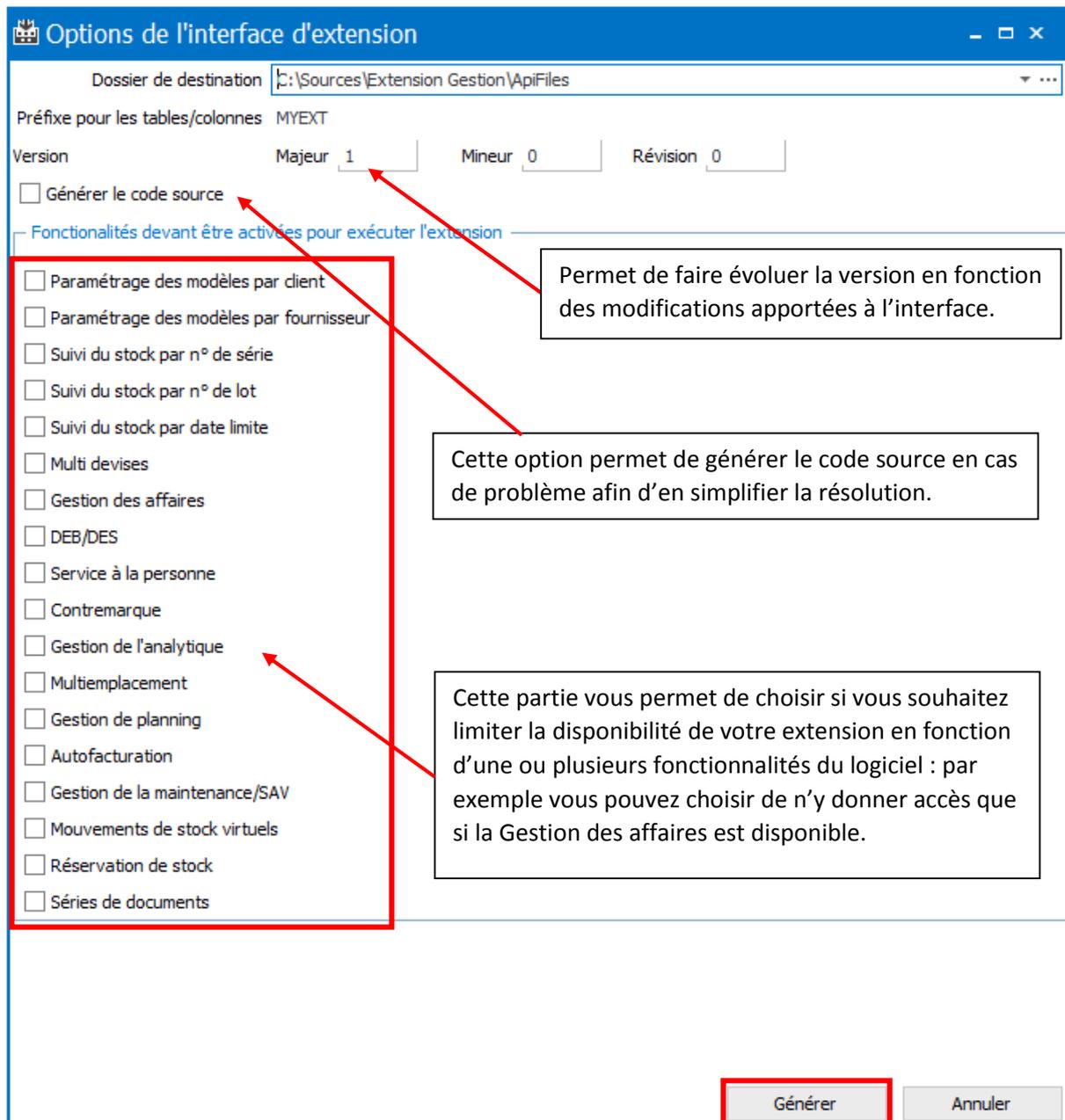


- Procéder de la même manière pour les différentes tables nécessaires au projet.



L'aide en ligne du logiciel contient de nombreuses informations. Elle vous aidera à déterminer les choix à faire lors de la création de vos tables personnalisées.

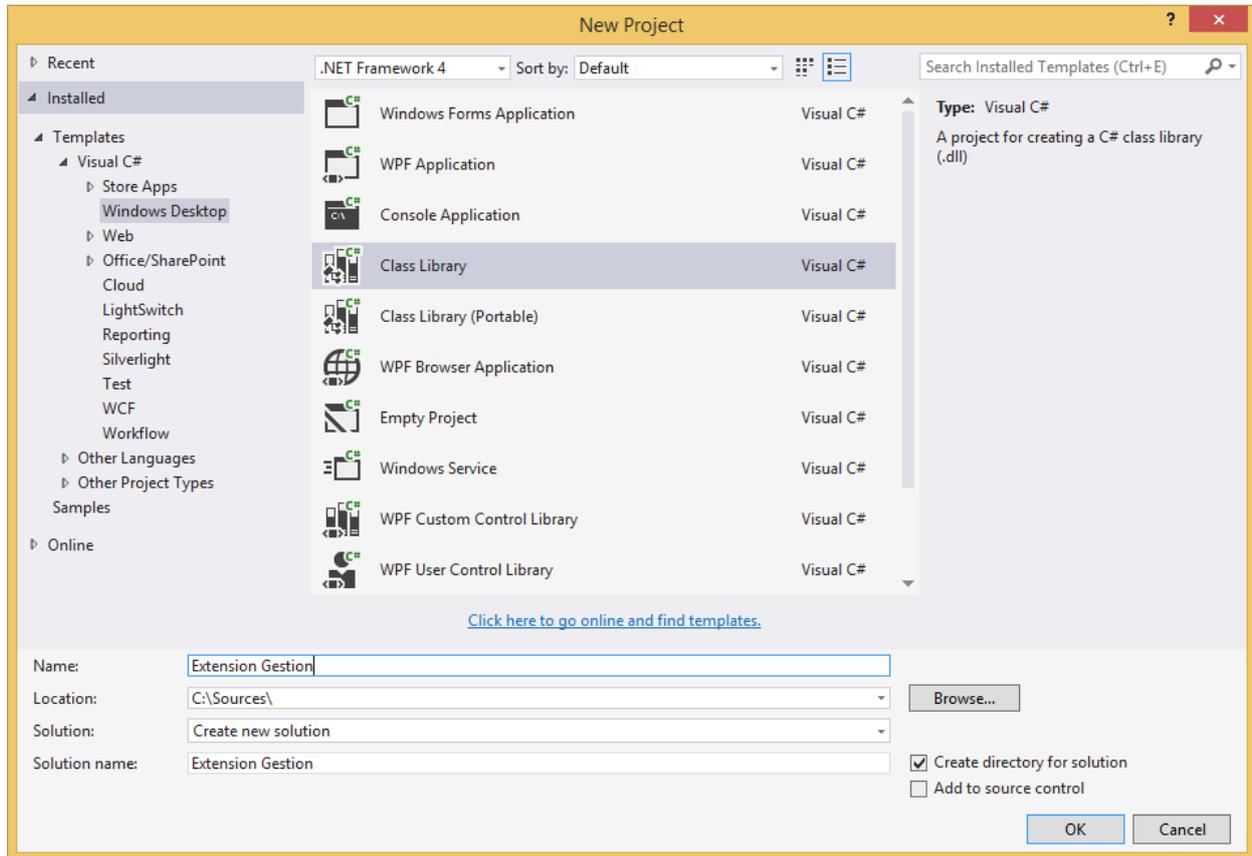
10. Pour générer l'interface, cliquer sur le menu *Développeur / Générer les interfaces*. Dans la fenêtre *Options de l'interface d'extension*, sélectionner le dossier de destination (de préférence un répertoire dédié) puis cliquer sur le bouton **Générer**.



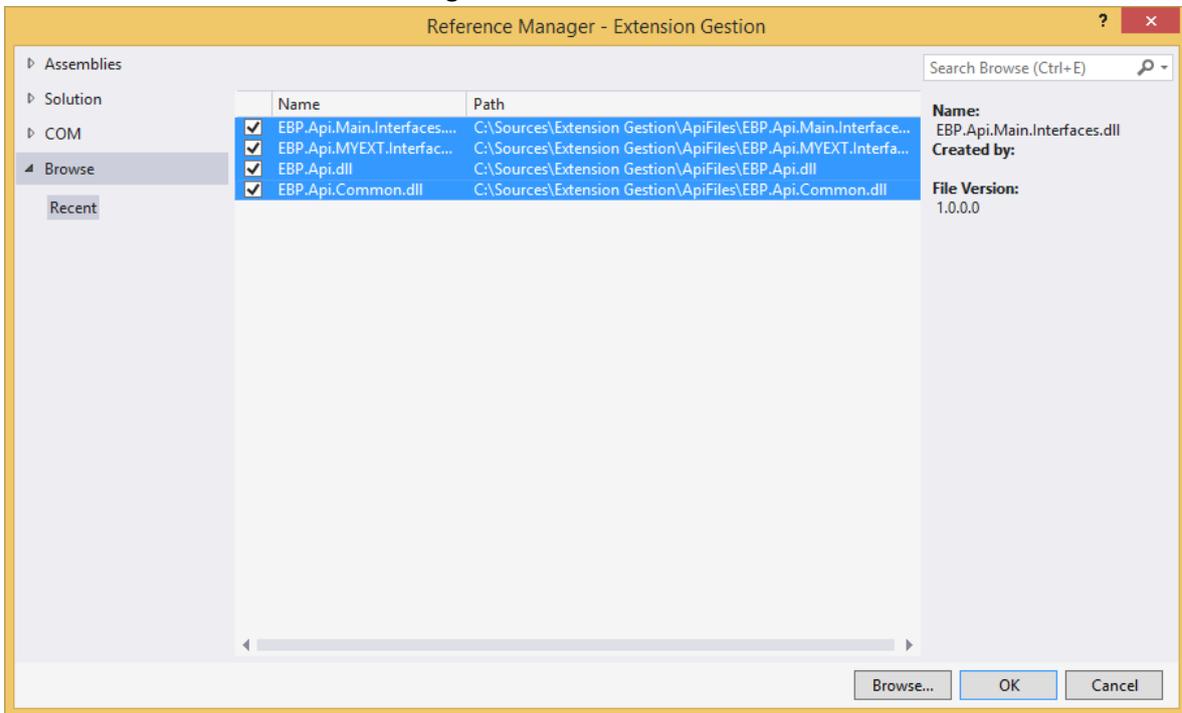
11. Une fois la génération terminée, fermer le dossier. Dans le menu *Développeur*, choisir *Sortir du mode Développeur* puis quitter l'application.

Etape 2 : Ouverture du projet d'extension

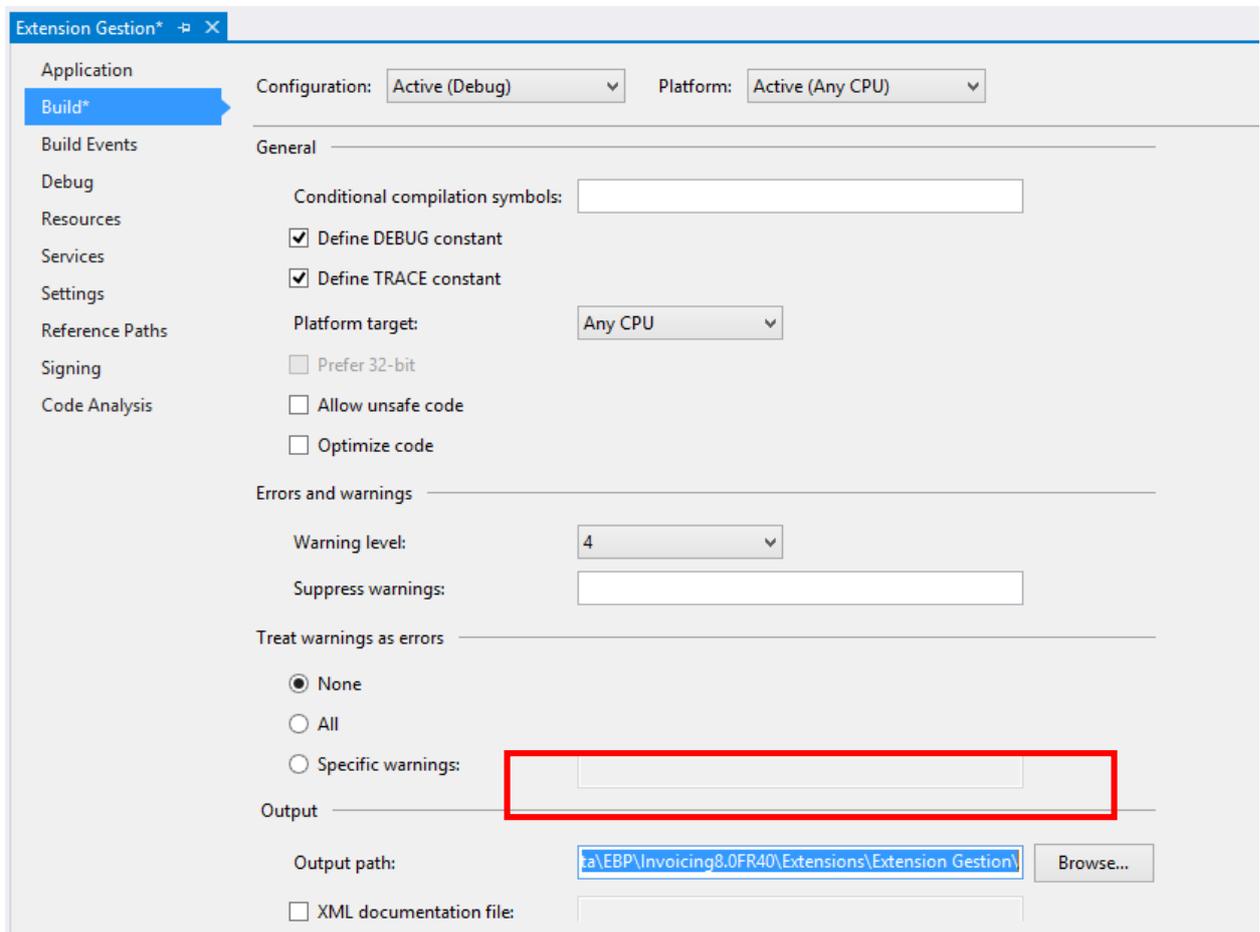
1. Dans **Visual Studio**, créer un projet de type *Class Library.NET Framework* version 4.0 ou supérieure nommé Extension Gestion.



2. Référencer les dll générées précédemment, disponibles dans le dossier de destination indiqué lors de la génération de l'interface.



3. Modifier le répertoire de sortie du projet en « C:\ProgramData\EBP\Invoicing8.0FR40\Extensions\Extension Gestion »



4. Créer une classe pour l'extension

- Créer une classe dérivant de la classe Extension : les propriétés et les méthodes évoquées ensuite sont indispensables.

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
    }
}
```

- Surcharger les propriétés abstraites :
 - ApiInterfaceLinkType

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Gets the type of the API interface ApiInterfaceLink generated class.
        /// </summary>
        /// <remarks>Return the type EBP.Api.Interfaces.ApiInterfaceLink generated in the
interface dll EBP.Api.XXXXXX.Interfaces.dll</remarks>
        protected override Type ApiInterfaceLinkType
        {
            get { return typeof(ApiInterfaceLink); }
        }
    }
}
```

- Description : description de l'extension

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Gets the extension description.
        /// </summary>
        /// <value>The extension description</value>
        protected override string Description
        {
            get { return "Extension pour la Gestion Commerciale"; }
        }
        ...
    }
}
```

- ExtensionId : Identifiant de l'extension (**cette valeur ne doit pas être modifiée une fois votre extension utilisée !**)

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Gets the extension Id. Must be unique for each extension.
        /// </summary>
        /// <value> The extension Id </value>
        /// <remarks> Do not change this value otherwise it may cause unpredictable behavior
in the application. </remarks>
        protected override Guid ExtensionId
        {
            get { return new Guid("43FF83AB-9F86-495E-9D13-356181639333"); }
        }
        ...
    }
}
```

- Name : Nom de l'extension

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Gets the extension name.
        /// </summary>
        /// <value>The extension name</value>
        protected override string Name
        {
            get { return "Extension Gestion"; }
        }
        ...
    }
}
```

- Surcharger les méthodes
 - ConfirmUserDefinedSchemaObjectDelete : cette méthode doit être surchargée mais ne sera pas détaillée ici.

```
using EBP.Api.Extension;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Gets whether a schema object can be safely deleted
        /// </summary>
        /// <param name="tableName">Table name</param>
        /// <param name="columnName">Column name. Null if it's a delete of a table</param>
        /// <returns>
        /// True for confirm this delete, otherwise false
        /// </returns>
        protected override bool ConfirmUserDefinedSchemaObjectDelete(string tableName,
            string columnName)
        {
            return true ;
        }
        ...
    }
}
```

- OnInitialized : permet d'initialiser toutes les entités et les fenêtres de l'extension

```
namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Initializes the extension. Override this method to register all extensions
        /// entities, forms, etc.
        /// </summary>
        /// <param name="errors">Errors to fill in if initialization fails</param>
        /// <returns><c>true</c> if extension initialization succeed, otherwise
        /// <c>false</c></returns>
        protected override bool OnInitialized(IErrors errors)
        {
            if (!base.OnInitialized(errors))
                return false;
            RegisterExtensionUserDefinedMenu(typeof(IContactFunctionEntryForm),
                "Fonction Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            RegisterExtensionUserDefinedMenu(typeof(IContactServiceEntryForm), "Service
            Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            RegisterExtensionUserDefinedMenu(typeof(ISubFonctionContactEntryForm),
                "Sous-Fonction Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            return true;
        }
        ...
    }
}
```

- Voici au final le code obtenu :

```

using EBP.Api.Extension;
using EBP.Api.Interfaces;
using EBP.Api.Interfaces.Invoicing.Module.Windows;
using EBP.Api.Interfaces.Misc;
using EBP.Api.Interfaces.UserDefinedForm;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Extension_Gestion
{
    public class ExtensionGestion : Extension
    {
        /// <summary>
        /// Initializes the extension. Override this method to register all extensions
        /// entities, forms, etc.
        /// </summary>
        /// <param name="errors">Errors to fill in if initialization fails</param>
        /// <returns><c>true</c> if extension initialization succeed, otherwise
        /// <c>false</c></returns>
        protected override bool OnInitialized(IErrors errors)
        {
            if (!base.OnInitialized(errors))
                return false;
            RegisterExtensionUserDefinedMenu(typeof(IContactFunctionEntryForm),
                "Fonction Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            RegisterExtensionUserDefinedMenu(typeof(IContactServiceEntryForm), "Service
            Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            RegisterExtensionUserDefinedMenu(typeof(ISubFonctionContactEntryForm),
                "Sous-Fonction Contact",
                new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
            return true;
        }

        /// <summary>
        /// Gets whether a schema object can be safely deleted
        /// </summary>
        /// <param name="tableName">Table name</param>
        /// <param name="columnName">Column name. Null if it's a delete of a table</param>
        /// <returns>
        /// True for confirm this delete, otherwise false
        /// </returns>
        protected override bool ConfirmUserDefinedSchemaObjectDelete(string tableName,
        string columnName)
        {
            return true;
        }

        /// <summary>
        /// Gets the extension name.
        /// </summary>
        /// <value>The extension name</value>
        protected override string Name
        {
            get { return "Extension Gestion"; }
        }
    }
}

```

```
    /// <summary>
    /// Gets the extension Id. Must be unique for each extension.
    /// </summary>
    /// <value>The extension Id</value>
    /// <remarks>Do not change this value otherwise it may cause unpredictable behavior
in the application.</remarks>
    protected override Guid ExtensionId
    {
        get { return new Guid("43FF83AB-9F86-495E-9D13-356181639333"); }
    }

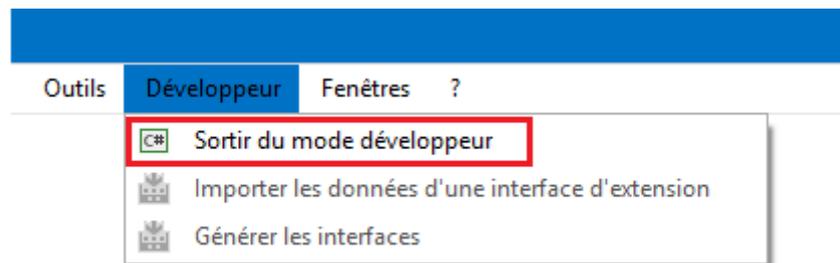
    /// <summary>
    /// Gets the extension description.
    /// </summary>
    /// <value>The extension description</value>
    protected override string Description
    {
        get { return "Extension pour la Gestion Commerciale"; }
    }

    /// <summary>
    /// Gets the type of the API interface ApiInterfaceLink generated class.
    /// </summary>
    /// <remarks>Return the type EBP.Api.Interfaces.ApiInterfaceLink generated in the
interface dll EBP.Api.XXXXXX.Interfaces.dll</remarks>
    protected override Type ApiInterfaceLinkType
    {
        get { return typeof(ApiInterfaceLink); }
    }
}
}
```

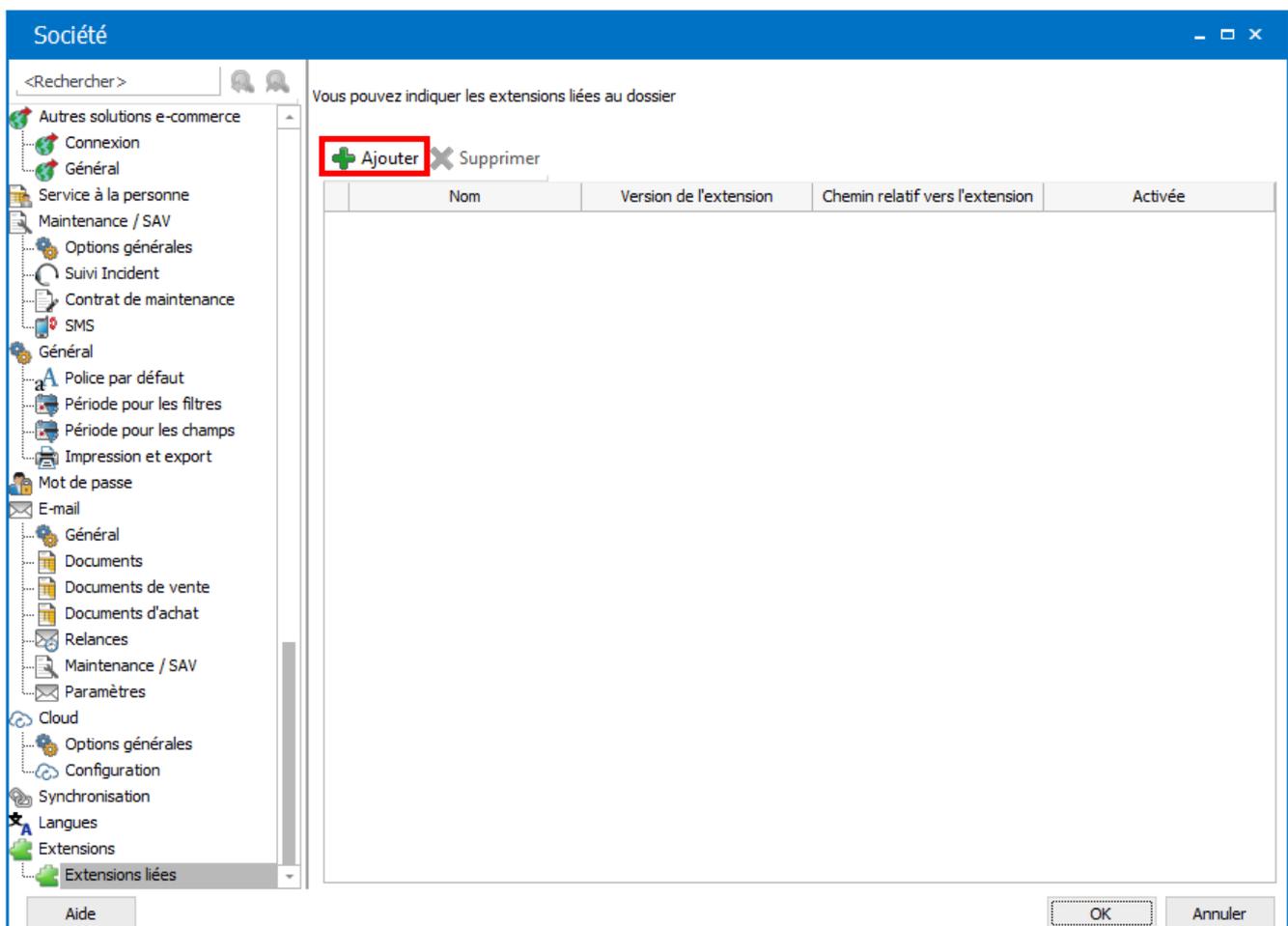
5. Compiler le projet. En cas de soucis à la compilation, vérifier si l'application EBP Gestion Commerciale Open Line est bien fermée. Dans le cas contraire, fermer l'application puis relancer la compilation du projet.

Etape 3 : Ajouter l'extension à un dossier de l'application EBP Gestion Commerciale Open Line

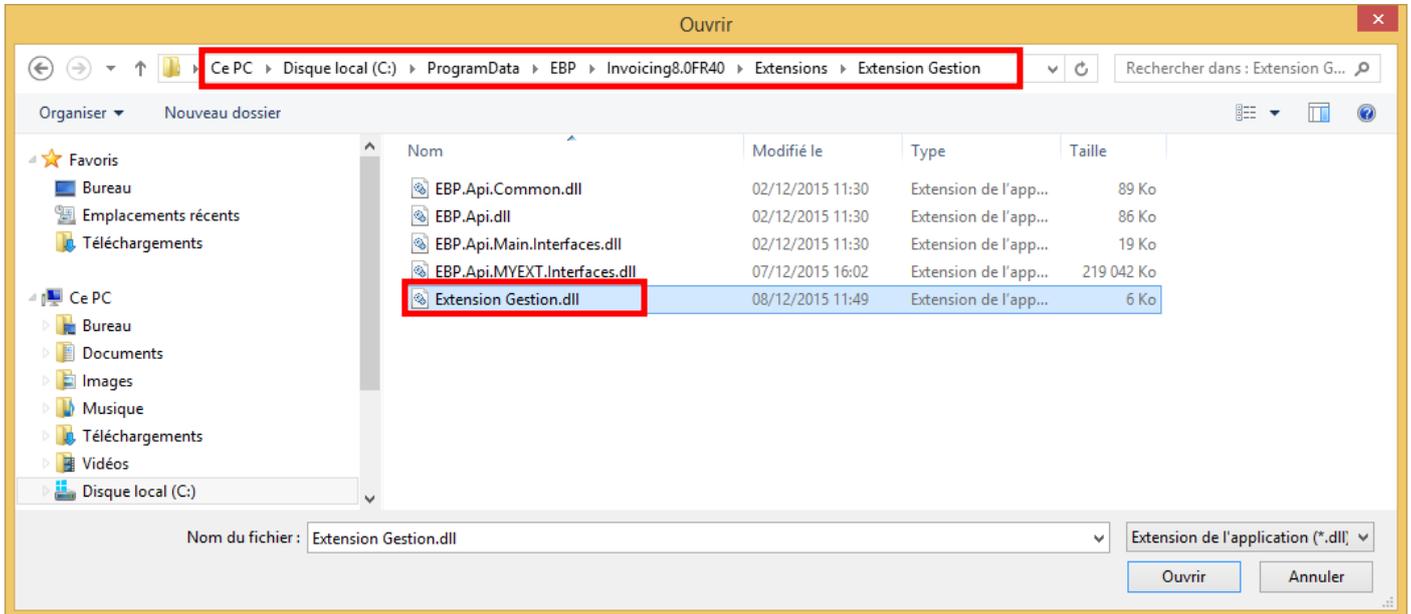
1. Vérifier que vous n'êtes pas en mode développeur



2. Créer un nouveau dossier ou ouvrir un dossier existant
3. Aller dans le menu *Paramètres / Société / Extensions liées*
4. Cliquer sur le bouton *Ajouter* pour ajouter votre extension dans votre dossier



5. Sélectionner la dll générée par le projet Visual Studio, puis cliquer sur Ouvrir.



La fenêtre suivante apparaît :

Vous pouvez indiquer les extensions liées au dossier

+ Ajouter ✕ Supprimer

Nom	Version de l'extension	Chemin relatif vers l'extension	Activée
Extension Gestion	1.0.0.0	Extension Gestion\Extension...	<input checked="" type="checkbox"/>
Extension pour la Gestion Commerciale			

Dans la colonne *Nom*, on retrouve le Nom de l'extension.

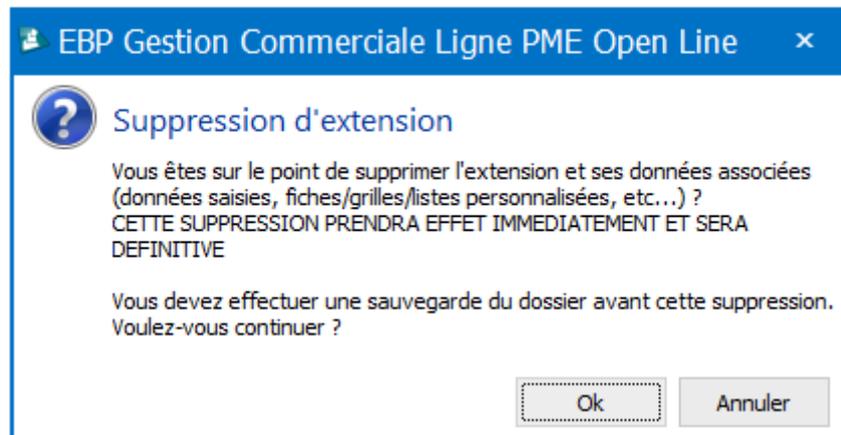
Dans *Version*, la version de votre projet d'extension.

La zone *Chemin relatif vers l'extension*, pointe vers le chemin où se trouve la dll importée.

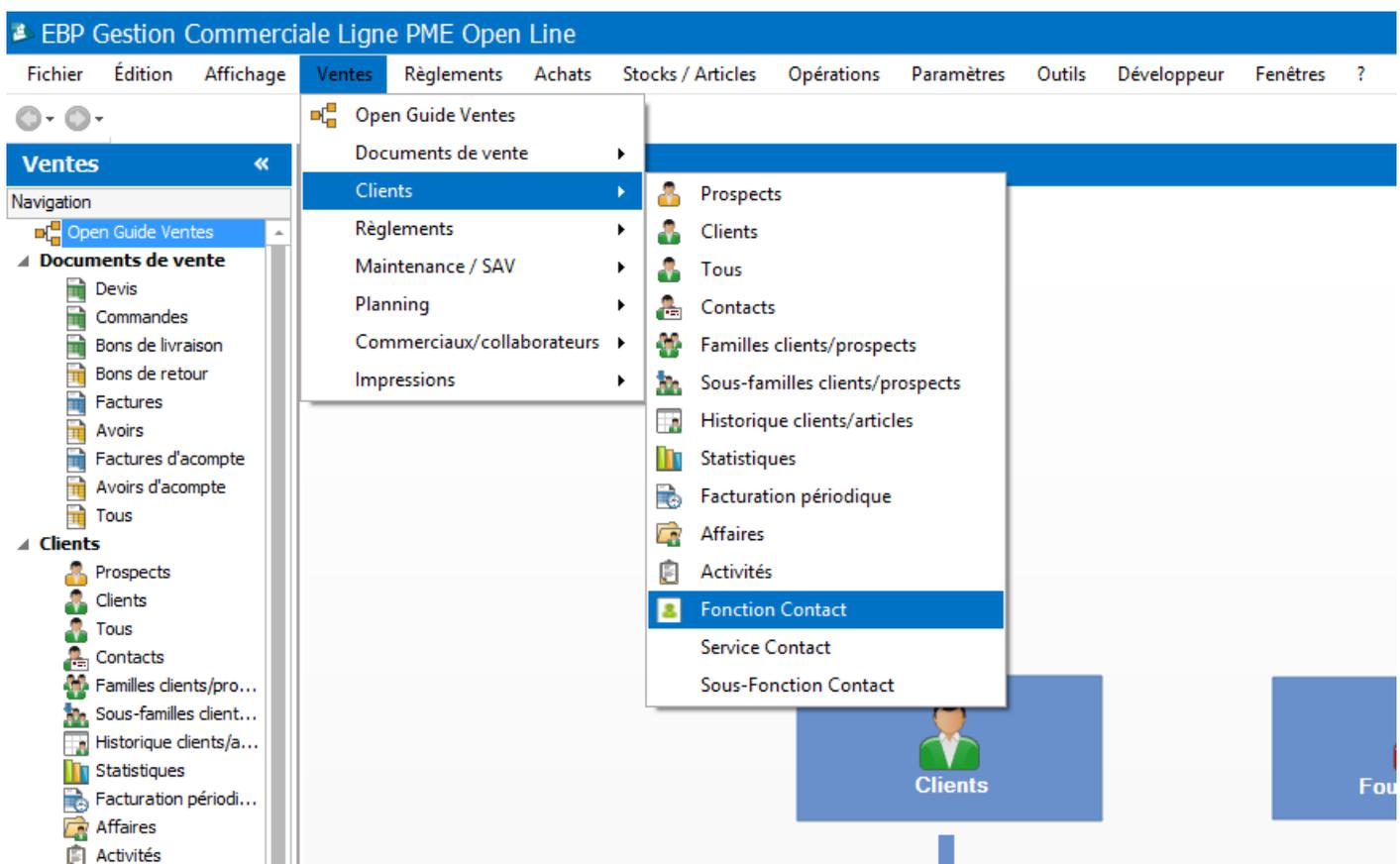
La case *Activée*, permet de voir si l'extension est appliquée. Si elle est décochée, les nouvelles fiches créées par l'extension ne seront pas visible dans l'application.

Aide OK Annuler

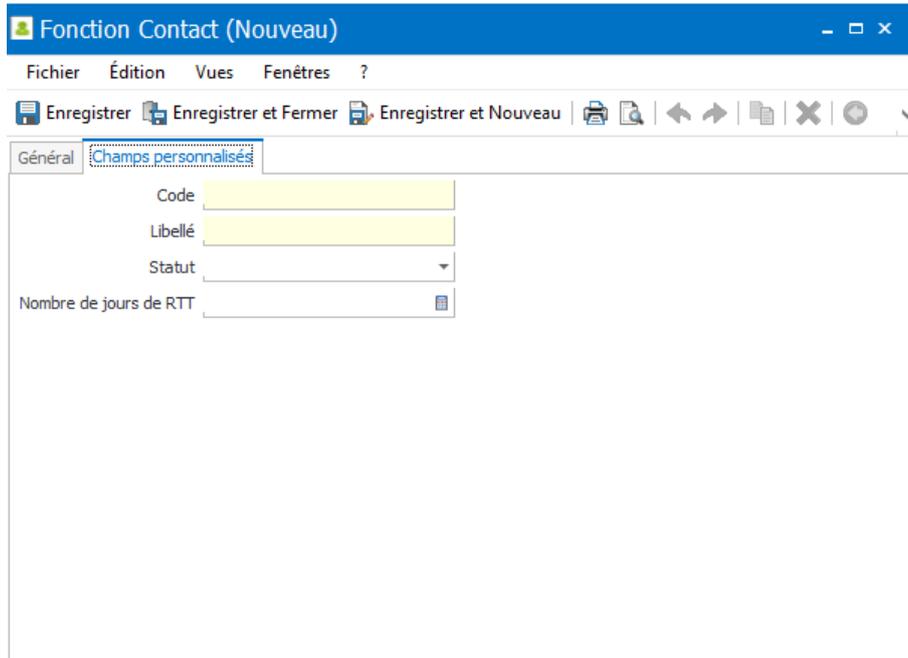
⚠ Avertissement : Le bouton Supprimer sert à supprimer l'extension. Vous pourrez la réimporter après l'avoir supprimée, mais **les données en rapport avec celle-ci seront définitivement perdues**. Un message d'avertissement est affiché lors de l'utilisation de ce bouton.



6. Fermer les options en cliquant sur OK et rouvrir le dossier.
7. Cliquer sur le menu Ventes / Clients : les tables créées sont disponibles.



8. Voici la fiche Fonction Contact :



 La personnalisation des fiches, pour les mettre en forme selon vos souhaits, sera à effectuer en mode Développeur, dans le dossier permettant de générer votre extension, afin que toutes les informations soient intégrées dans celle-ci.