Projet API étendu

Version bêta - 12/2015

Table des matières

Descriptif

Ce projet a pour objectif de vous donner un aperçu de la mise en place des EXTENSIONS qui vont étendre le comportement et l'apparence de la fiche Contact de la Gestion Commerciale. Il illustre également certaines fonctionnalités avancées des API.

Trois nouvelles tables personnalisées sans table maître, *ContactFunction* (Fonction de contact), *SubContactFunction* (Sous-fonction de contact) et *ContactService* (Service de contact) ont été ajoutées dans le dossier API.

Pour chacune de ces tables, un champ de type liste déroulante a été ajouté dans la table Contact. Une nouvelle vue de la fiche Contact a permis de remplacer les zones de saisie texte Fonction et Service par des lookups.

Une règle métier simple a été définie dans la classe *ContactEntityExtension* (extension de l'entité Contact) : saisie d'une sous-fonction de contact possible seulement si fonction de contact a été renseignée.

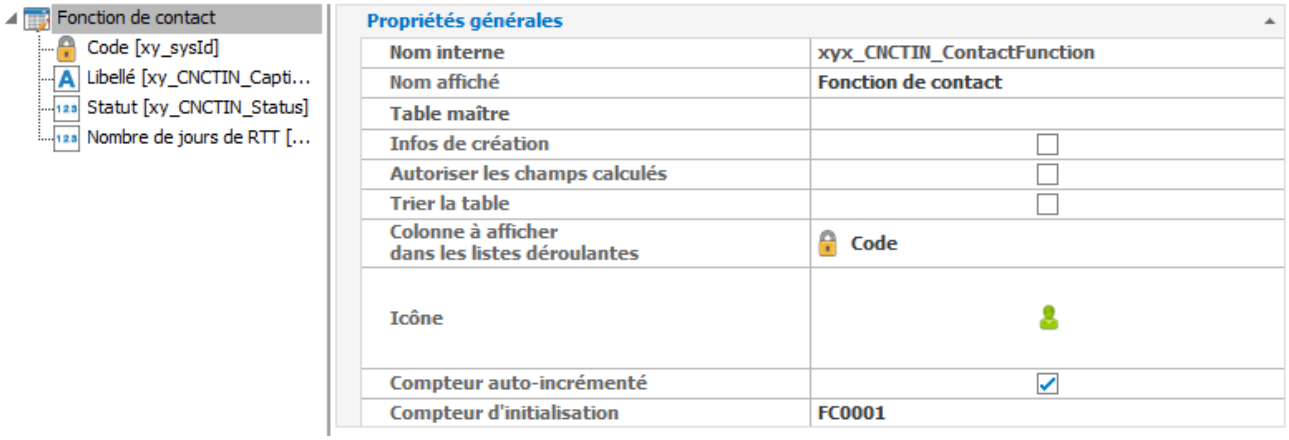
L'extension de la fiche Contact (classe *ContactEntryFormExtension*) illustre l'utilisation d'un filtre contextuel (classe *SubFunctionLookupContextualFilter*) permettant le filtrage des enregistrements affichés dans le lookup des sous-fonctions de contact en fonction de l'enregistrement sélectionné dans le lookup des fonctions. Sur cette extension de fiche, une action permettant d'ouvrir la fiche du service de contact sélectionné a été ajoutée.

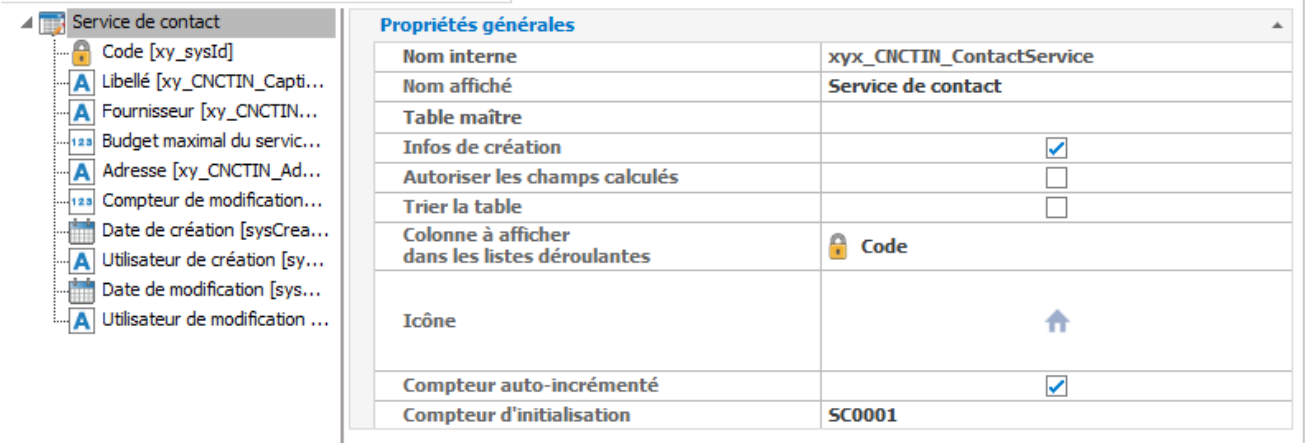
La fiche Service de contact doit disposer d'un détail listant les équipes du service. Pour cela, une table *ContactServiceTeam* (Equipe de service de contact) a été définie dans le dossier API avec comme table maître la table *ContactService*. Les extensions des entités associées à ces deux tables sont les classes *ContactServiceEntityExtension* et *ContactServiceTeamEntityExtension*.

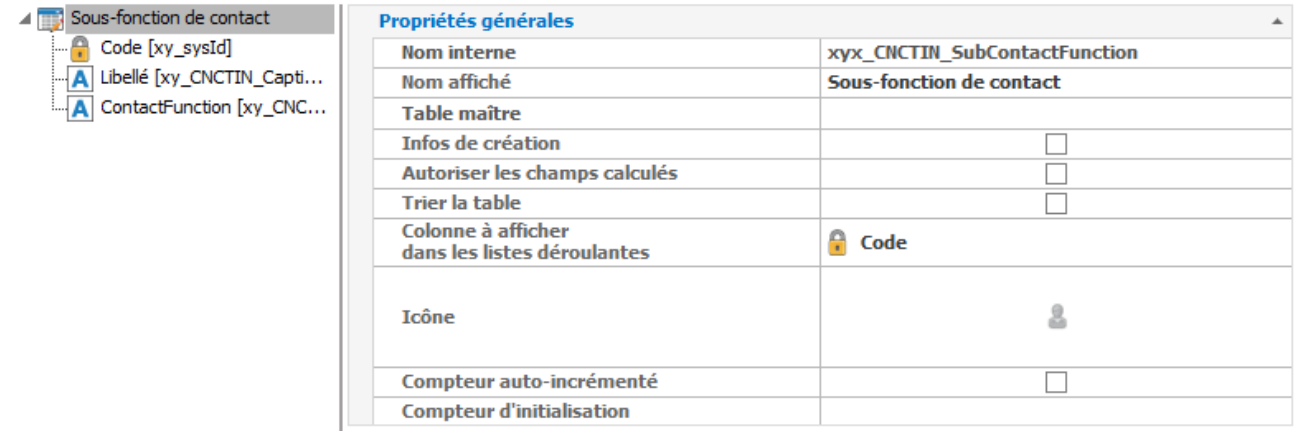
Des règles métiers sont définies dans ces deux extensions afin de détecter si le total du budget des équipes dépasse le budget alloué au service. Cette détection est effectuée lors de la modification d'une équipe (OnChanging), après la modification d'une équipe (*OnChanged*) et lors de la suppression d'une équipe (*OnDeletedFromParentCollection*).

Si le budget d'un service est dépassé, le texte des éléments de la liste des équipes est en rouge sur la fiche des Services. On utilise l'évènement *CustomDrawNodeCell* du contrôle de la liste.

Tables créées par le sample (elles correspondent aux entités ajoutées)







Etape 1 : Ouverture du Projet dans Visual Studio

A l'ouverture du projet, vous vous retrouverez avec des classes déjà créées.

Dossier contenant toutes les classes Entities présentes dans votre API

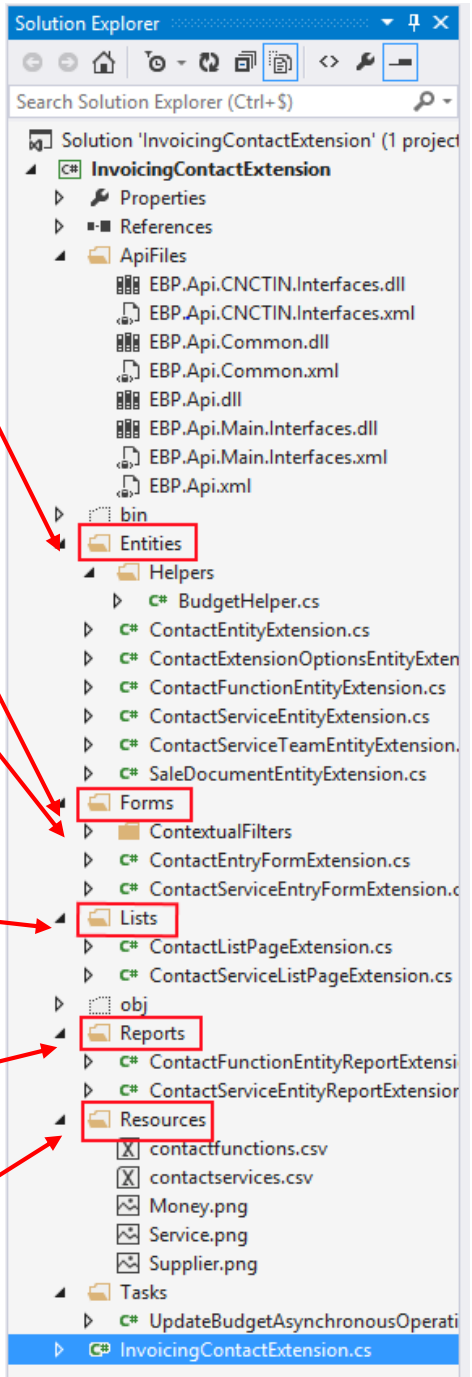
Dossier contenant toutes les Extensions des Forms présentes dans votre API. Elles vous permettent de définir de nouveaux comportements visuels sur le formulaire.

Dossier contenant les filtres contextuels qui s'appliquent sur l'EntryForm d'une entité. Dans cet exemple, le filtre s'applique sur l'EntryForm de ma classe Contact

Dossier dans lequel se trouvent les extensions des ListPages

Dossier dans lequel sont stockés les paramétrages des fenêtres d'impression et qui permettent de définir les membres qui sont associés au filtrage (indispensable).

Dossier contenant différentes ressources (les icones, fichiers .csv)

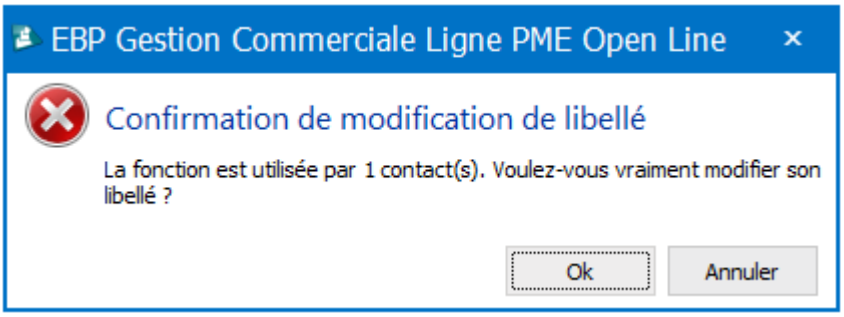


Important : Pour chaque table créée, il est impératif d'ajouter une classe ReportEntity.

Etape 2 : Déclaration de l'Entity

La classe *ContactFunctionEntityExtensions.cs* permet de regrouper toutes les règles métiers appliquées à l'entité *Fonction Contact.*
Dans notre exemple, nous allons essayer de savoir si la fonction d'un contact est en cours de changement, puis nous récupèrerons la valeur par défaut du RTT selon le statut sélectionné.

La méthode *OnChanging* permet d'agir avant la modification d'une fonction contact déjà créée. Elle nous permet de déterminer si la fonction peut être affectée à un contact ou non, en fonction de son état (modifié ou créé).



Si la réponse est OK, les modifications apparaitront dans la fiche du contact auquel est appliquée la fonction.

```
public classe ContactFunctionEntityExtension : EntityExtension<IContactFunctionEntity>
 {
   #region Fields

   private const int DefaultManagerRttDayCount = 16;
   private const int DefaultEmployeeRttDayCount = 12;
   #endregion

   #region Methods

   /// <summary>
   /// Méthode opérant lors du changement d'un membre
   /// </summary>
   /// <param name="args">Informations on changes</param>
   protected override void OnChanging(EBP.Api.Interfaces.Entity.IChangingEventArgs args)
   {
     base.OnChanging(args);
     if (args.Cancel)
       return;

     // Vérification de la présence d'un changement
     if (Equals(args.EntityMember, Entity.CaptionMember))
     {
       // Demande une confirmation si la fonction est affectée à un contact
      if (Entity.Mode != EntityMode.Append) // si tel est le cas
       {
         // Query database
         // SELECT COUNT(*) FROM Contact WHERE xy_CNCTIN_ContactFunction = Entity.sysId
         string query = string.Format("SELECT COUNT(*) FROM {0} WHERE {1} = '{2}'",
         ContactSchemaTable.ContactTableName,
         ContactSchemaTable.CNCTIN_ContactFunctionColumnName, Entity.sysId);
         try
         {
             int contactUsingFunctionCount =
             (int)InvoicingContactExtension.Instance.Database.ExecuteScalar(query);
           if (contactUsingFunctionCount > 0)
           {
             // Crée un nouveau message args
             IShowMessageEventArgs messageArgs =
             Utils<InvoicingContactExtension>.GetInterface<IShowMessageEventArgs>();
              messageArgs.Action = ShowMessageAction.OKCancel;
             messageArgs.Content = string.Format("La fonction est utilisée par {0} contact(s).
             Voulez-vous vraiment modifier son libellé ?",
                contactUsingFunctionCount);
              messageArgs.DefaultAction = ShowMessageResult.OK;
              messageArgs.Instruction = "Confirmation de modification de libellé";
              // Affiche le message de confirmation (uniquement dans un EntryForm)
              Entity.ShowMessage(messageArgs);
              if (messageArgs.Result == ShowMessageResult.Cancel)
              {
                 // Supprime la modification
                args.Cancel = true;
                // si aucune erreur est ajoutée à args.Errors, on set la valeur d'args.Handled à
                vrai
                 args.Handled = true;
              }
           }
         }
         catch (Exception ex)
         {
           // Construit une erreur quand une exception est causé par une requête SQL
```

```
            IDatabaseException databaseException =
            Utils<InvoicingContactExtension>.Catch<IDatabaseException>(ex);
          if (databaseException == null)

           throw;
          args.Errors.Add(ErrorKind.Error, ex.Message);
          args.Cancel = true;
        }
      }
    }
}

/// <summary>
/// Méthode permettant d'obtenir la valeur par default du RTT en fonction du statut du contact
/// sélectionné (DefaultManagerRTTDayCount est défini à 16 par défaut)
/// </summary>
/// <param name="statusValue">The status value.</param>
/// <returns>The default RTT days count</returns>
/// <exception cref="InvalidOperationException">Valeur de statut non définie</exception>
private int GetDefautRttCountByStatus(int statusValue)
{
  switch (statusValue)
  {
    case ContactFunction_StatusType.Cadre:
      return DefaultManagerRttDayCount;
    case ContactFunction_StatusType.Employe:
    case ContactFunction_StatusType.Agent_de_maitrise:
      return DefaultEmployeeRttDayCount;
    default:
      throw new InvalidOperationException("Valeur de statut non définie");
  }
}


/// <summary>
/// Relève l'évènement suivant le changement d'un membre
/// </summary>
   /// <param name="args">The <see cref="EBP.Api.Interfaces.Entity.IChangedEventArgs"/> instance
   containing the event data.</param>
protected override void OnChanged(EBP.Api.Interfaces.Entity.IChangedEventArgs args)
{
  base.OnChanged(args);
  // Check if member is Status member (use object.Equals to compare interface implementations)
  if (Equals(args.EntityMember, Entity.StatusMember))
  {
   // Si la valeur RttDaysCount correspond à la valeur par défault selon l'ancienne valeur du
   statut, set la nouvelle valeur RTTDaysCount par défault pour le nouveau statut
    if (Entity.Status.HasValue)
    {
      bool changeRttDaysCountValue = false;
      if (Entity.StatusMember.PreviousValue.HasValue && Entity.RttDaysCount.HasValue)
      {


        // si le statut avait une valeur par défault, on applique un changement si
        RTTDaysCount n'avait pas de valeur par défault
        int previousDefaultValue =
        GetDefautRttCountByStatus(Entity.StatusMember.PreviousValue.Value);
                changeRttDaysCountValue = previousDefaultValue == Entity.RttDaysCount.Value;
      }
      else if (!Entity.StatusMember.PreviousValue.HasValue && !Entity.RttDaysCount.HasValue)
```

```
        {
            // si le statut n'avait pas une valeur par défault, on applique un changement si
            RTTDaysCount n'avait pas de valeur par défault
          changeRttDaysCountValue = true;
        }
        if (changeRttDaysCountValue)


            Entity.RttDaysCount = GetDefautRttCountByStatus(Entity.Status.Value);
      }
    }
  }
         /// <summary>
        /// Methode appelée au moment de la sauvgarde de l'entité
        /// </summary>
        /// <param name="e">Cancel entity event args</param>
        protected override void OnSaving(EBP.Api.Interfaces.Entity.ICancelEntityEventArgs e)
        {
          base.OnSaving(e);
          if (e.Cancel)
            return;

          // Check unique constraint on Caption
           // SELECT COUNT(*) FROM xy_CNCTIN_Caption WHERE xy_CNCTIN_Caption = Entity.Caption AND
           xy_sysId <> Entity.sysId
           string query = string.Format("SELECT COUNT(*) FROM {0} WHERE {1} = '{2}' AND {3} <>
           '{4}'",
            CNCTIN_ContactFunctionSchemaTable.CNCTIN_ContactFunctionTableName,
            CNCTIN_ContactFunctionSchemaTable.CNCTIN_CaptionColumnName, Entity.Caption,
            CNCTIN_ContactFunctionSchemaTable.sysIdColumnName, Entity.sysId);
          try
          {
           int contactUsingFunctionCount =
           (int)InvoicingContactExtension.Instance.Database.ExecuteScalar(query);
            if (contactUsingFunctionCount > 0)
            {
                  Entity.Errors.Add(ErrorKind.Error, string.Format("Le libellé '{0}' est déjà
                  attribué", Entity.Caption));
              e.Cancel = true;
            }
          }
          catch (Exception ex)
          {
            // Remonte une erreur si l'exception est due à une requête sql
           IDatabaseException databaseException =
           Utils<InvoicingContactExtension>.Catch<IDatabaseException>(ex);
            if (databaseException == null)
              throw;
            Entity.Errors.Add(ErrorKind.Error, ex.Message);
            e.Cancel = true;
          }
        }
        #endregion
      }
    }
```

Etape 3 : Déclaration de l'EntryForm

La classe *ContactEntryFormExtension* est une extension des entryform (les formulaires) présentes dans la Gestion Commerciale.

Elle vous permet de surcharger des traitements sur le formulaire en question, par exemple :

- ➔ Ajouter un traitement à l'initialisation : `protected override void OnEntityInitialized()`
- ➔ Ajouter un filtre contextuel sur un lookup lorsque celui-ci est créé :
  `protected override IApiLookupContextualFilter OnLookupContextualFilterNeeded(IReadOnlyEntityMemberBase entityMember)`

- ➔ Gestionnaire d'évènement d'une action permettant de mettre à jour l'accessibilité du bouton lié à cette action
  `private void ShowServiceAction_ActionUpdate(object sender, EventArgs e)`
  Par exemple : Lorsque le bouton *Afficher les services* dans le formulaire Contact passe de l'état actif à desactivé en se basant sur les changement du contenu du lookup Service.

Déclaration des proprietés

```
/// <summary>
/// Extension du formulaire de la classe Contact de l'application Gestion Commerciale
/// </summary>
public classe ContactEntryFormExtension : EntryFormExtension<IContactEntryForm>
{
  #region Properties

  /// <summary>
  /// Récupération de l'icone
  /// </summary>
  private ImageList ImageList { get; set; }

  /// <summary>
  /// Récupération et définition du filtre sous-fonction utilisé pour filtrer le lookup sous-
  /// fonction
  /// </summary>
  private SubFunctionLookupContextualFilter SubFunctionFilter { get; set; }

  /// <summary>
  /// Récupération et définition de l'action Afficher les services
  /// </summary>
  /// </value>
  private IAction ShowServiceAction { get; set; }
  #endregion
```

Déclaration du constructeurs de la classe

```
#region Constructor

/// <summary>
/// Initialise une nouvelle instance de la classe <see cref="ContactListPageExtension"/>
/// </summary>
public ContactEntryFormExtension()
{
  ImageList = new ImageList();
  ImageList.Images.Add(Resources.Service);
}
#endregion
```

Déclaration des méthodes

Méthode *OnEntityInitialized()* appelée à la création d'une entité

```
/// <summary>
/// Appelée à la création d'une entité
/// </summary>
/// <remarks>
/// On peut surcharger cette méthode s'abonner aux évènements d'une entité
/// </remarks>
protected override void OnEntityInitialized()
{
  base.OnEntityInitialized();
  ContactEntityExtension contactExtension =
    EntryForm.Entity.GetExtension(InvoicingContactExtension.Id) as ContactEntityExtension;
  contactExtension.ContactFunctionChanged += contactExtension_ContactFunctionChanged;
}
```

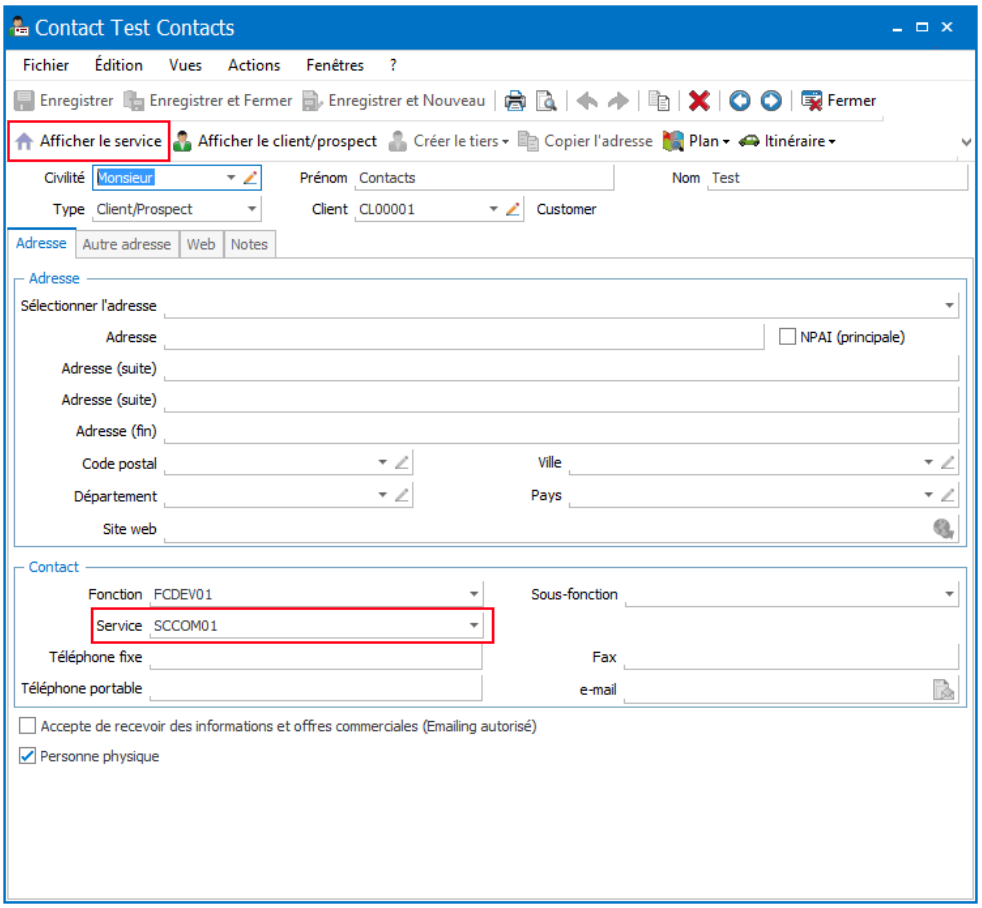
Méthode *ContactExtension_ContactFunctionChanged(),* permet de mettre à jour le filtre contextuel de la sous-fonction selon le changement de la fonction

```
private void contactExtension_ContactFunctionChanged(object sender, EventArgs e)
{
  // Mise à jour du filtre sous-fonction après le changement de la fonction du contact
  if (SubFunctionFilter != null)
    SubFunctionFilter.FunctionId = EntryForm.Entity.ContactFunction;
}
```

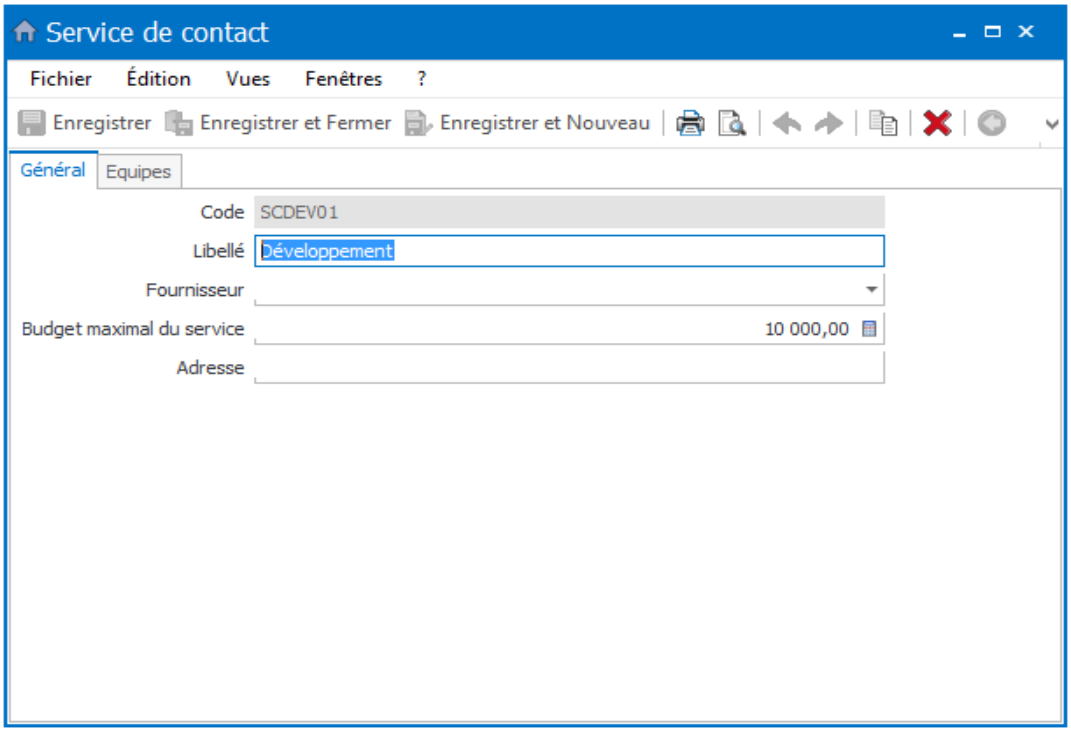
Méthode *OnLookupContextualFilterNeeded(),* qui est appelée lors de la création d'un lookup

```
    /// <summary>
    /// appelée lors de la création d'un lookup. Utile pour l'ajout d'un filtre sur ce lookup
    /// </summary>
    /// <param name="entityMember">entité pour laquelle le lookup est créé</param>
    /// <returns>
    /// filtre à appliquer sur ce control
    /// </returns>
    protected override IApiLookupContextualFilter
OnLookupContextualFilterNeeded(IReadOnlyEntityMemberBase entityMember)
    {
       // Vérifie si la méthode est appelée pour le lookup sous-fonction qui est lié
       // à un membre sous-fonction contact
      // (Use object.Equals to compare interface implementations)
      if (Equals(entityMember, EntryForm.Entity.SubContactFunctionMember))
      {
        // crée et retourne le filtre sous-fonction
        SubFunctionFilter = new SubFunctionLookupContextualFilter();
        return SubFunctionFilter;
      }
      return base.OnLookupContextualFilterNeeded(entityMember);
    }
```

Méthode *FillActions()*, on appelle cette méthode pour ajouter des actions à l'entryform . Par exemple, le bouton *Afficher le service* qui est ajouté à la fiche Contact.



Ce bouton vous permet d'afficher le service sélectionné dans le lookup Service.




```
/// <summary>
/// méthode appelée lorsque l'entryform à besoin de créer ses actions
/// </summary>
/// <param name="actions">Actions to fill</param>
protected override void FillActions(IActionCollection actions)
{
  base.FillActions(actions);
  // création de l'action ShowServiceAction (afficher le service)
  ShowServiceAction = Utils<InvoicingContactExtension>.GetInterface<IAction>();
  ShowServiceAction.ImageList = ImageList;
  ShowServiceAction.ImageIndex = 0;
  ShowServiceAction.Text = "Afficher le service";
  ShowServiceAction.RegisterRight(new Guid(UserDefinedGuids.ContactServiceRightGuid),
    StandardRightKind.Read, false);
  ShowServiceAction.ActionUpdate += ShowServiceAction_ActionUpdate;
  ShowServiceAction.ActionExecute += ShowServiceAction_ActionExecute;
  // Ajout d'une action
  actions.Add(ShowServiceAction);
}
```

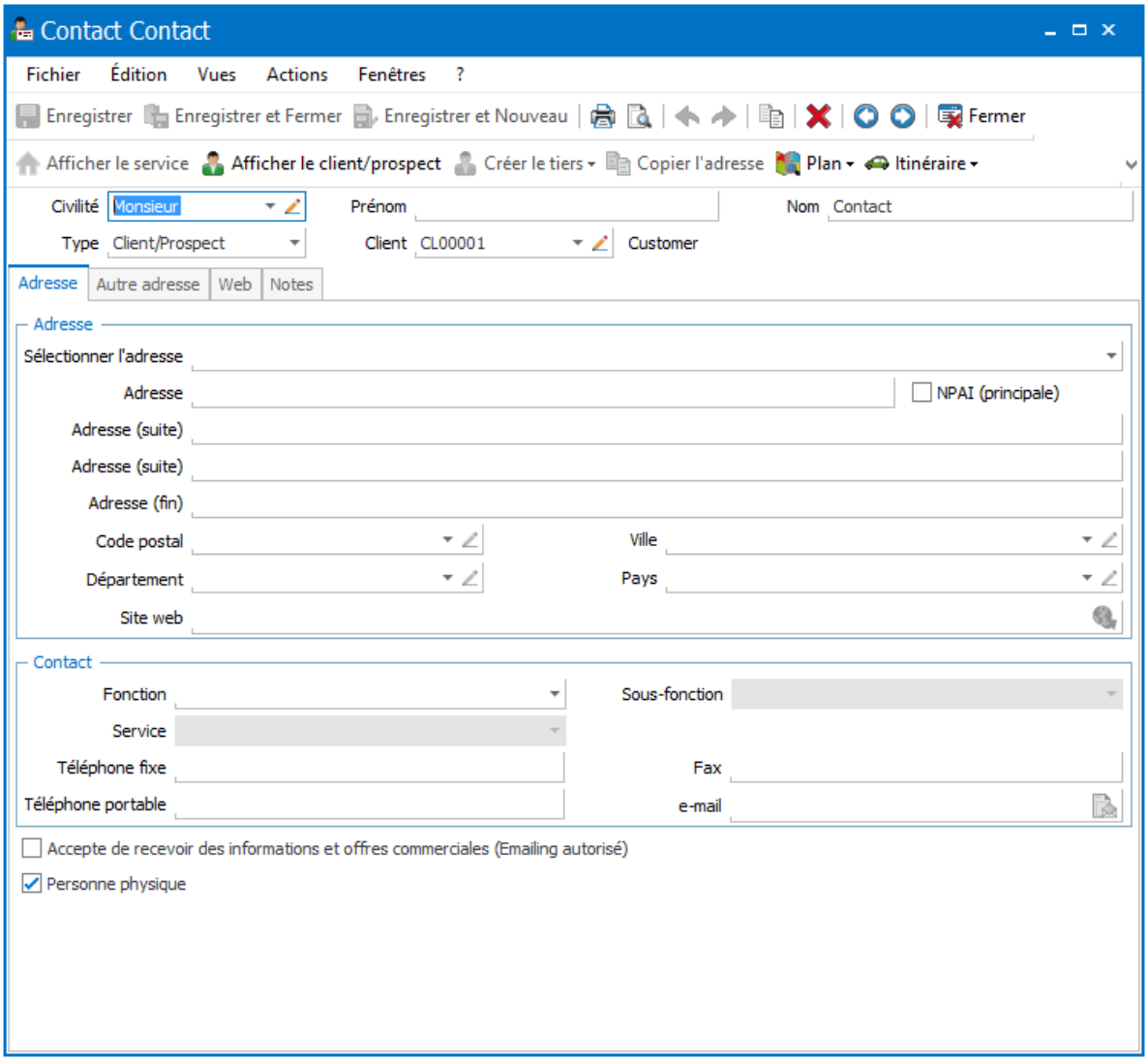
La méthode *OnDispose()* qu'il faut surcharger, permet d'effectuer les tâches de libération des ressources.

```
 protected override void Dispose()
  {
    base.Dispose();
    // libère tous les champs à libérer
    if (ImageList != null)
    {
      ImageList.Dispose();
      ImageList = null;
    }
    if (SubFunctionFilter != null)
    {
      SubFunctionFilter.Dispose();
      SubFunctionFilter = null;
    }
    if (ShowServiceAction != null)
    {
      ShowServiceAction.Dispose();
      ShowServiceAction = null;
    }
}
```

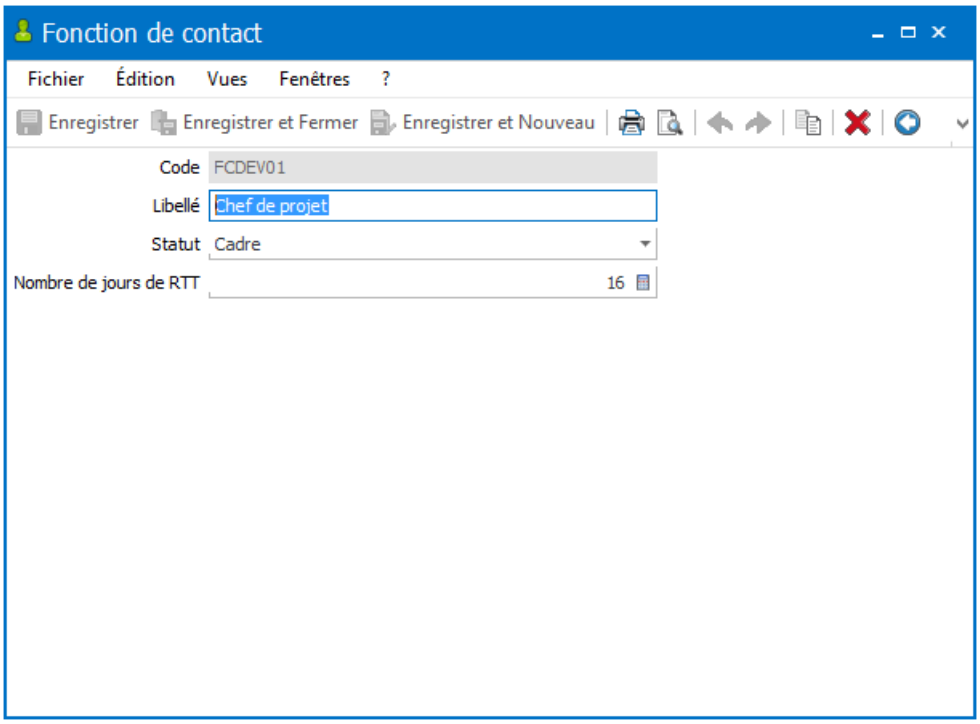
Etape 4 : Application d'un filtre contextuel

Dans notre exemple, un filtre contextuel s'applique sur un contact et permet de filtrer les sous-fonctions selon ce qui est saisi dans le lookup *Fonction*.

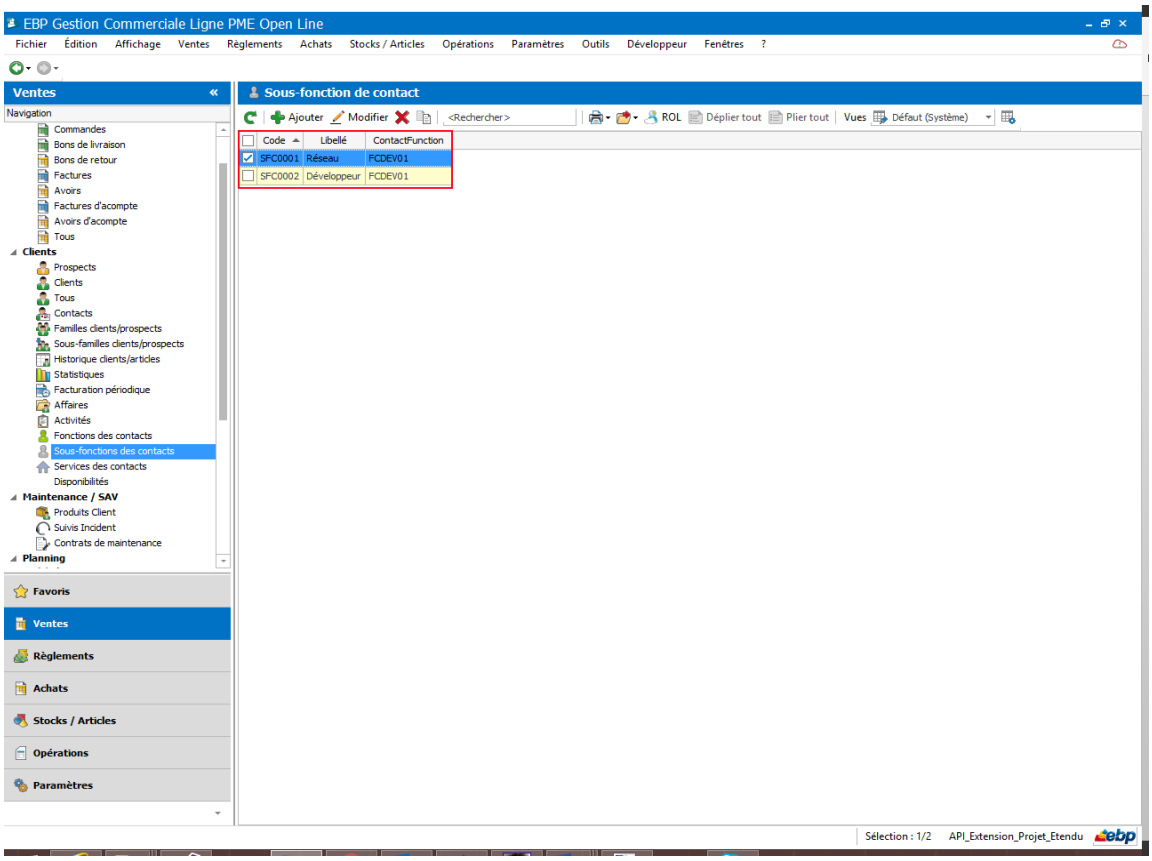
Par exemple :
Notre créons un contact qui s'appelle Monsieur *Contact*, lié à un client CL00001 *Client*



Dans le menu *Fonctions des contacts*, nous avons une fonction dont le libellé est Chef de Projet

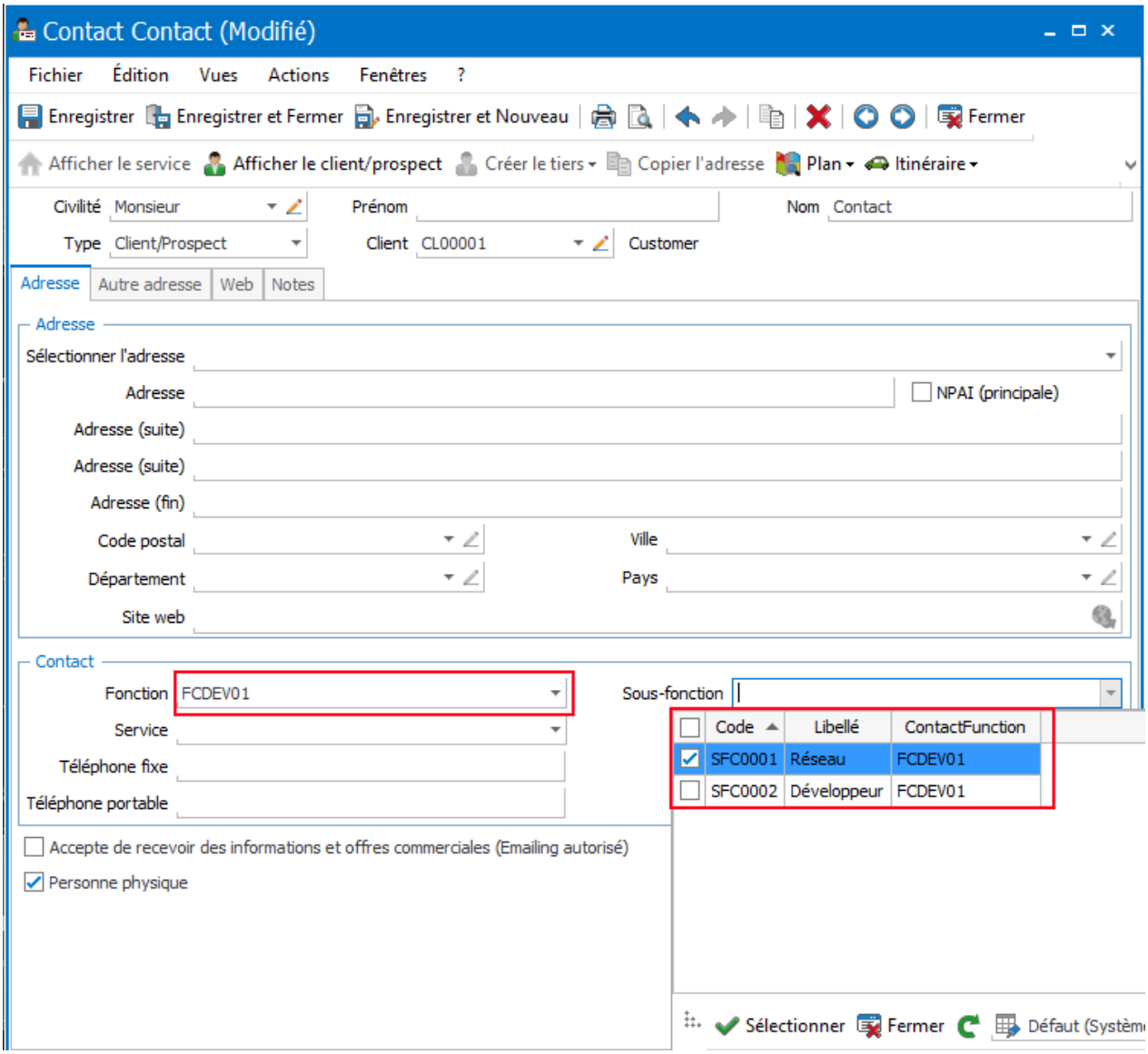


A notre chef de projet, nous allons ajouter deux sous-fonctions contacts *Développeur* et *réseau,* depuis le menu *Sous-fonctions* des contacts.



Nous allons appliquer au contact créé auparavant la fonction *chef de projet*.

En appliquant cette fonction, un filtre s'applique également sur le lookup *Sous-fonction* afin de proposer toutes les sous-fonctions liées à la fonction *Chef de projet*



Dans la fenêtre *solution explorer* de Visual Studio, vous disposez de la classe *SubFunctionLookupContextualFilter* dans le Forms/ContextualFilters, qui vous permettra de voir le code utilisé pour obtenir ce fonctionnement.

```
namespace InvoicingContactExtension.Forms.ContextualFilters
{
  /// <summary>
  /// Filtre appliqué au formulaire Contacts qui permet de filtrer les sous-fonctions
  /// selon la fonction sélectionnée.
  /// </summary>
  public class SubFunctionLookupContextualFilter : LookupContextualFilterBase
  {
    private string functionId;

    /// <summary>
    /// Récupère et définit l'identifiant de la fonction sélectionnée dans le lookup
    /// qui va filtrer les sous-fonctions
    /// </summary>
    public string FunctionId
    {
      get { return functionId; }
      set
      {
        if (functionId != value)
        {
          functionId = value;

          /// appel de la fonction OnFiltersCreating() crée ci-dessous
          OnFiltersChanged();
        }
      }
    }

    /// <summary>
    /// fonction appelée lorsque le filtre est créé
    /// méthode à utiliser pour créer des filtres
    /// </summary>
    protected override void OnFiltersCreating()
    {
      base.OnFiltersCreating();
      // On crée un filtre basé sur le FunctionId
      CreateFilter(CNCTIN_SubContactFunctionSchemaTable.CNCTIN_SubContactFunctionTableName,
       CNCTIN_SubContactFunctionSchemaTable.CNCTIN_ContactFunctionColumnName,
       FilterOperator.Equal, FunctionId);
    }
  }
}
```

Etape 5 : Initialisation des menus

Dans la classe *InvoicingContactExtension,* qui hérite de la classe *Extension (InvoicingContactExtension : Extension)*, il est possible d'initialiser les menus nécessaires pour accéder aux nouvelles tables.

Pour commencer, il faut surcharger les propriétés abstraites :

Surcharge des proprieté abstraites

ApiInterfaceLinkedType : qui permet de retourner le type d'interfaces de l'API concernées par le projet.

```
protected override Type ApiInterfaceLinkType
   {
     get { return typeof(EBP.Api.Interfaces.ApiInterfaceLink); }
   }
```

Description : retourne la description de l'extension.

```
protected override string Description
   {
      get { return "Extension permettant de définir des services et des fonctions pouvant
      être associés aux contacts."; }
   }
```

ExtensionId : retourne l'identifiant de l'extension (qui se doit d'être unique pour chaque extension).

```
protected override Guid ExtensionId
   {
     get { return Id; }
   }
```

Name : retourne le nom de l'extension.

```
protected override string Name
   {
     get { return "Extension des contacts"; }
   }
```

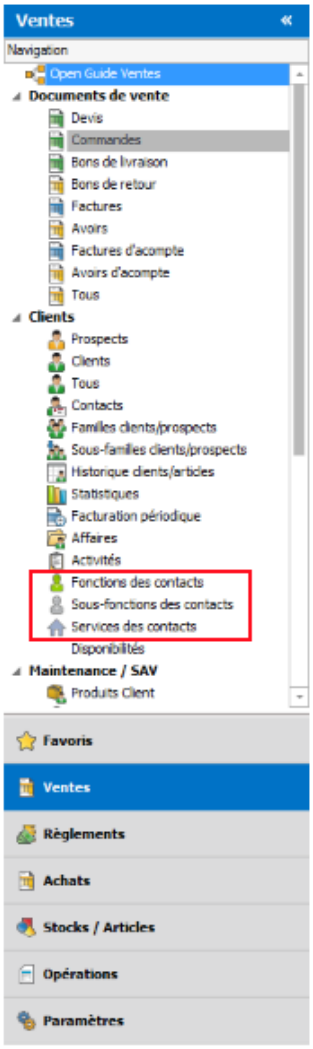
Méthode initialisation

Par la suite, il est nécessaire d'initialiser l'extension.

Pour cela, il faut overrider la method OnInitialized() pour lister toutes les entités et forms présentes dans l'extension.

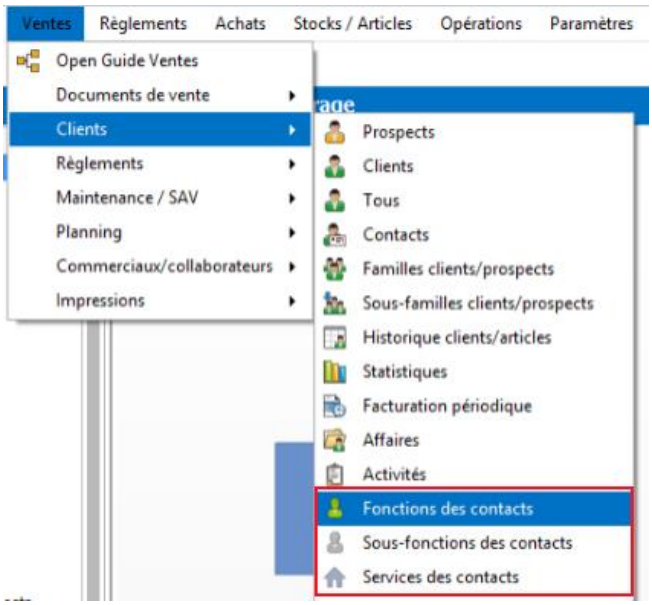
```
protected override bool OnInitialized(IErrors errors)
   {
     if (!base.OnInitialized(errors))
       return false;

     // Initialize HostOwnerForm
     HostOwnerForm = HostOwner;

     // Set static instance
     Instance = this


// RegisterEntityExtention
// Initialise une nouvelle instance de la classe EBP.Api.Extension.Extension
// ContactFunctionEntityExtension Classe regroupant toutes les règles de gestion pour la
// fonction contact
RegisterEntityExtension<ContactFunctionEntityExtension>();

// ContactServiceEntityExtension Classe regroupant toutes les règles de gestion pour
// Service Contact

RegisterEntityExtension<ContactServiceEntityExtension>();

RegisterEntityExtension<ContactServiceTeamEntityExtension>();

// Methode ContactEntityExtension utilisé pour personnaliser le comportement
// de l'entité contact
RegisterEntityExtension<ContactEntityExtension>();

// SaleDocumentEntityExtension permet de définir le comportement du document de vente
// Elle prend en paramètre le type de document de vente
RegisterEntityExtension<SaleDocumentEntityExtension<IQuoteSaleDocumentEntity>>();
RegisterEntityExtension<SaleDocumentEntityExtension<IOrderSaleDocumentEntity>>();

// Permet d'avoir accès aux options de l'application EBP, qui regroupe toutes
// Les options spécifiques
RegisterOptionExtension<ContactExtensionOptionsEntityExtension>();

// Permet de faire des Imports/Export des tables maîtres utilisées dans l'API
// les interfaces des Entités se trouvent dans l'espace de nom UserDefinedEntity et sont
// préfixées par un « I »
RegisterImportCategoryExtension<IContactFunctionEntity>("ContactFunction");
RegisterImportCategoryExtension<IContactServiceEntity>("ContactService");

// Permet de faire des Imports/Export des tables maîtres utilisées dans l'API
RegisterReportExtension<ContactFunctionEntityReportExtension>(new
Guid(Guids.CustomerReportGroup));
RegisterReportExtension<ContactServiceEntityReportExtension>(new
Guid(Guids.CustomerReportGroup));

// Enregistre les nouveaux comportements du listpage associé au GUID
RegisterListPageExtension<ContactServiceListPageExtension>(new
Guid(UserDefinedGuids.ContactServiceListInfoGuid));
RegisterListPageExtension<ContactListPageExtension>(new Guid(ListPagesGuids.Contact));

// Enregistre l'extension d'une entryForm
RegisterEntryFormExtension<ContactServiceEntryFormExtension>();
RegisterEntryFormExtension<ContactEntryFormExtension>();
```

```
        // Ajout des différents tables créées dans le menu
        // typeof pour définir la table à ajouter associée aux GUIDS
        // MenuGuids pour lier les tables dans un menu particulier
        // (ici dans Documents de Ventes)
        // NavBarGuids pour lier les tables au menu NavBar
        RegisterExtensionUserDefinedMenu(typeof(IContactFunctionEntity), "Fonctions des contacts",
        new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
        RegisterExtensionUserDefinedMenu(typeof(ISubContactFunctionEntity), "Sous-fonctions des
        contacts",
        new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));
        RegisterExtensionUserDefinedMenu(typeof(IContactServiceEntity), "Services des contacts",
        new Guid(MenuGuids.SaleCustomer), new Guid(NavBarGuids.SalesCustomer));

        // Enregistre les tâches asynchrones
        RegisterAsynchronousOperationExtension<UpdateBudgetAsynchronousOperation>(false);

        return true;
}
```



Une fois que la .dll est installée, on constate que des menus sont apparus dans la NavBar (Barre de navigation à gauche de l'écran)

Mais également dans le menu du haut



Dans la méthode OnRunned (IErrors errors), vous pouvez déclarer le traitement nécessaire lors que l'extension est lancée (retourne false si l'extension n'est pas prête à être lancée)

```
protected override bool OnRunned(IErrors errors)
   {
     if (!base.OnRunned(errors))
       return false;

     // Import/Export les fonctions et services depuis les ressources si les tables sont vides
     IGenericImportService service = GetService<IGenericImportService>();
     IErrors importErrors = Utils<InvoicingContactExtension>.GetInterface<IErrors>();
     string formatQuery = "SELECT COUNT(*) FROM {0}";
      string query = string.Format(formatQuery,
      CNCTIN_ContactFunctionSchemaTable.CNCTIN_ContactFunctionTableName);
     if ((int)Database.ExecuteScalar(query) == 0)
     {
       string importDataFilePath = string.Format("{0}.csv", Path.GetTempFileName());
       File.WriteAllText(importDataFilePath, Resources.contactfunctions, Encoding.Default);
       // "ContactFunctionDefaultImport" représente le nom des paramètres d'importations
       // créées depuis le mode API
      service.Import(importDataFilePath, "ContactFunction", "ContactFunctionDefaultImport",
      GenericImportErrorAction.Cancel,
          importErrors);
     }
```

```
      query = string.Format(formatQuery,
      CNCTIN_ContactServiceSchemaTable.CNCTIN_ContactServiceTableName);
     if ((int)Database.ExecuteScalar(query) == 0)
     {
       string importDataFilePath = string.Format("{0}.csv", Path.GetTempFileName());
       File.WriteAllText(importDataFilePath, Resources.contactservices, Encoding.Default);
       // "ContactServiceDefaultImport" représente le nom des paramètres d'importations
       // créées depuis le mode API
       service.Import(importDataFilePath, "ContactService", "ContactServiceDefaultImport",
      GenericImportErrorAction.Cancel,
          importErrors);
     }
     // ici les options sont importés
      If
      (ContactExtensionOptionsEntityExtension.Instance.MustLaunchUpdateBudgetAsynchronousTaskOnRun.
      Value == true)
      {
       // Lance les tâches asynchrones si elles sont mise en place dans les options
       AsynchronousRegisteredOperations[UpdateBudgetAsynchronousOperation.Guid].StartTriggers();
      }
     return true;
   }
```

Méthode *OnStopped(),* elle est appelée lorsque l'extension est stoppée

```
protected override void OnStopped()
   {
     base.OnStopped();
     Instance = null;
   }
```

Méthode *OnDispose(),* effectue la libération des ressources de l'extension

```
protected override void OnDisposed()
   {
     if (!IsDisposed)
     {
       Instance = null;
     }
   }
```

Etape 6 : Initialisation de l'Entity Report

Concernant les EntityReports (Par exemple la classe *ContactFunctionEntityReportExtension qui se trouve dans le dossier « Reports »*), elles héritent de la Classe ReportEntityExtension.

Ces classes permettent de stocker les paramétrages des fenêtres d'impression et de définir les membres associés au filtrage.

```
public classe ContactFunctionEntityReportExtension : ReportEntityExtension<IContactFunctionEntity>
```

La classe ContactFunctionEntityReportExtension s'applique sur l'entité de la table Fonction de contact.

Dans cette classe, vous trouverez une déclaration des champs présents dans la table maître (en l'occurence la table *Fonction de contact*).

```
/// <summary>
/// The code lower bound of the filter settings.
/// </summary>
private IStringEntityMember codeFrom;
/// <summary>
/// The code upper bound of the filter settings.
/// </summary>
private IStringEntityMember codeTo;
/// <summary>
/// The caption lower bound of the filter settings.
/// </summary>
private IStringEntityMember captionFrom;
/// <summary>
/// The caption upper bound of the filter settings.
/// </summary>
private IStringEntityMember captionTo;
/// <summary>
/// The boolean indicating if executive status must be get.
/// </summary>
private IBoolEntityMember executiveStatus;
/// <summary>
/// The boolean indicating if employee status must be get.
/// </summary>
private IBoolEntityMember employeeStatus;
/// <summary>
/// The boolean indicating if agent status must be get.
/// </summary>
private IBoolEntityMember agentStatus;
```

Vous trouverez également une méthode *CreateMembers()* qui comme son nom l'indique permet de créer les membres associés au filtrage.

```
protected override void CreateMembers()
   {
     base.CreateMembers();
            CreateFromToMembers<IStringEntityMember,
      IStringEntityMember>(EntityDefinition.sysIdMember,  out codeFrom, out codeTo);
     codeFrom.Nullable = true;
     codeTo.Nullable = true;
            CreateFromToMembers<IStringEntityMember,
      IStringEntityMember>(EntityDefinition.CaptionMember, out captionFrom, out captionTo);
     captionFrom.Nullable = true;
     captionTo.Nullable = true;
      CreateMemberAuthorizedValue<int?>(EntityDefinition.StatusMember, ref employeeStatus,
      ContactFunction_StatusType.Employe);
     CreateMemberAuthorizedValue<int?>(EntityDefinition.StatusMember, ref agentStatus,
     ContactFunction_StatusType.Agent_de_maitrise);
     CreateMemberAuthorizedValue<int?>(EntityDefinition.StatusMember, ref executiveStatus,
     ContactFunction_StatusType.Cadre);
   }
```