


Traduction de l'article: L'utilisation de MAPI Extended au sein de vos applications

par Louis-Guillaume MORAND ([Page perso de Louis-Guillaume MORAND](#))

Date de publication : 04/09/2008

Dernière mise à jour :

Traduction de l'article  **Bridging the gap between .NET and Extended MAPI** présentant l'utilisation d'Extended MAPI pour mieux contrôler les applications comme Microsoft Outlook.

Introduction.....	3
0-A - Bénéfices.....	3
0-B - Drawbacks.....	3
0-C - Background.....	3
I - Préparation de la solution.....	3
I-A - Pré-requis.....	4
I-B - Créer une solution.....	4
II - Le truc Outlook.....	5
III - La bibliothèque C++ mixte.....	7
IV - Ajout de fonctionnalités.....	7
IV-A - Le fichier d'entête.....	7
IV-B - Initialisation et nettoyage.....	10
V - The usage.....	11
VI - Lire l'entête.....	13
VII - Lire le nom du profil Outlook courant.....	15
VIII - Définit la couleur du label de rendez-vous.....	18
IX - Notes.....	21

Introduction

Dans cet article, vous allez apprendre comment implémenter une librairie C++ qui aidera à utiliser les fonctions Extended MAPI qui ne sont pas disponibles via Outlook Object Model. Dans le passé, vous deviez acheter une bibliothèque supplémentaire ou deviez utiliser la technologie COM et créer cette DLL native par vous-même. Une autre façon est de réinventer la roue et de définir toutes les interfaces du côté .Net. Ceci vous autorisera d'utiliser ces interfaces directement depuis le C# ou VB.Net en triant les structures .Net et fonctions et structures non managées. C'est alors que vient une superbe bibliothèque C++ avec dans laquelle vous pouvez aisément mélanger du code manage et du code natif dans un unique environnement.

0-A - Bénéfices


- Toutes les interfaces et fonctions sont déjà définies dans le SDK Windows
- Nous n'avons pas besoin d'exposer une interface COM
- Aucun composant externe ne doit pas être déployé ou enregistré sur les systèmes cibles
- Classes et interfaces .NET exposées publiquement
- Aucune classes wrapper nécessaires pour les composants COM externes
- Entièrement intégré dans votre solution, avec le support du Debug

0-B - Drawbacks

- Une connaissance du C++ est requise
- Une connaissance des interfaces Extended MAPI est requise

0-C - Background

Le Modèle Outlook Object Model (OOM) est puissant et fournit l'accès à plusieurs fonctionnalités qui utilisent et manipulent les données stockées dans Outlook et Exchange Server. Pourtant, chaque développeur Outlook sérieux arrive à un point où il a besoin de certaines fonctionnalités, propriétés ou champs qui ne sont pas accessibles via le OOM ou qui sont bloqués par des restrictions de sécurité. Dans le passé, vous deviez utiliser une bibliothèque COM externe très réputée comme:

-  **Redemption** de Dmitry Streblechenko (la documentation vivante d'Extended MAPI)
- **Security Manager** d'Add-In-Express.com

Pendant que ces bibliothèques sont très puissantes et flexibles car sont utilisables depuis tout langage de programmation, elles sont néanmoins des bibliothèques externes qui ont besoin d'être déployées et enregistrées avec votre application. Ceci peut parfois être problématique. Quand vous utilisez Extended MAPI depuis .NET, il existe une autre bibliothèque nommée:

- **MAPI33** de Sergey Golovkin

I - Préparation de la solution

Avant que vous ne commenciez à utiliser Extended MAPI, vous devez installer certains éléments obligatoires sur votre machine de développement.

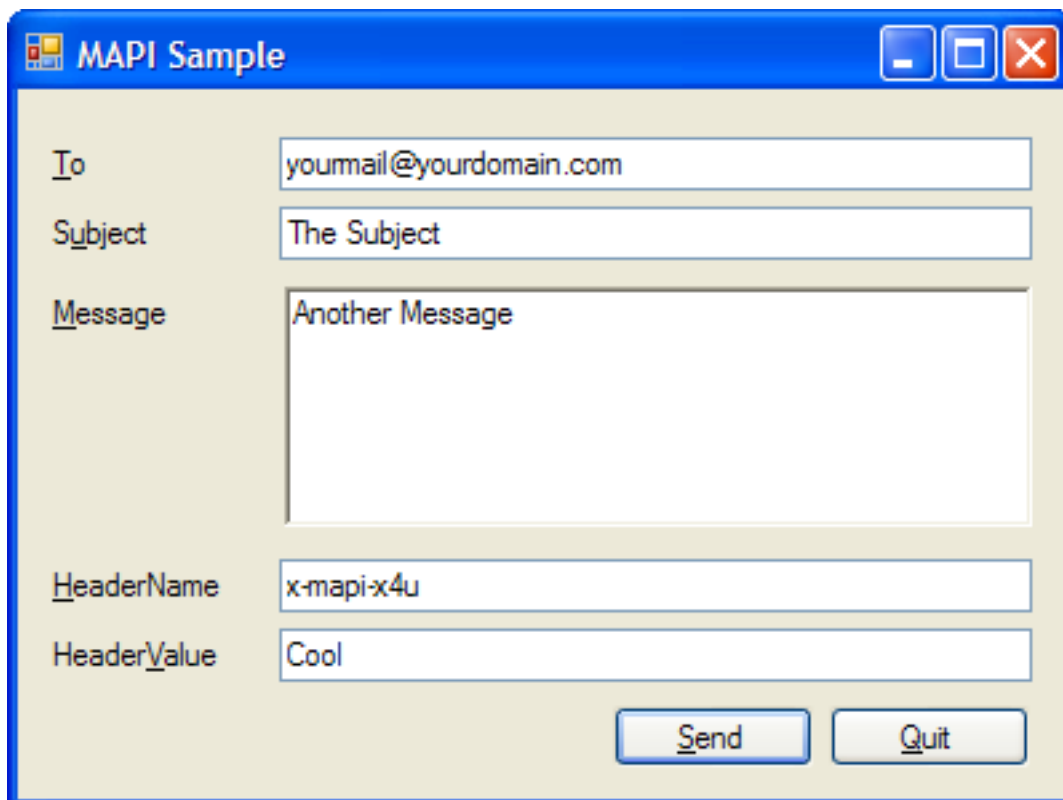
I-A - Pré-requis

- **Microsoft Outlook 2003 / Outlook 2007**
- Les PIAs (Primary Interop Assemblies) Office **2003 / 2007**
- **Microsoft Visual Studio 2005**
- **Windows Platform SDK**

Note: Si vous avez déjà installé une ancienne version du SDK Microsoft Exchange Server 5.5, vous n'avez pas besoin de la plateforme SDK.

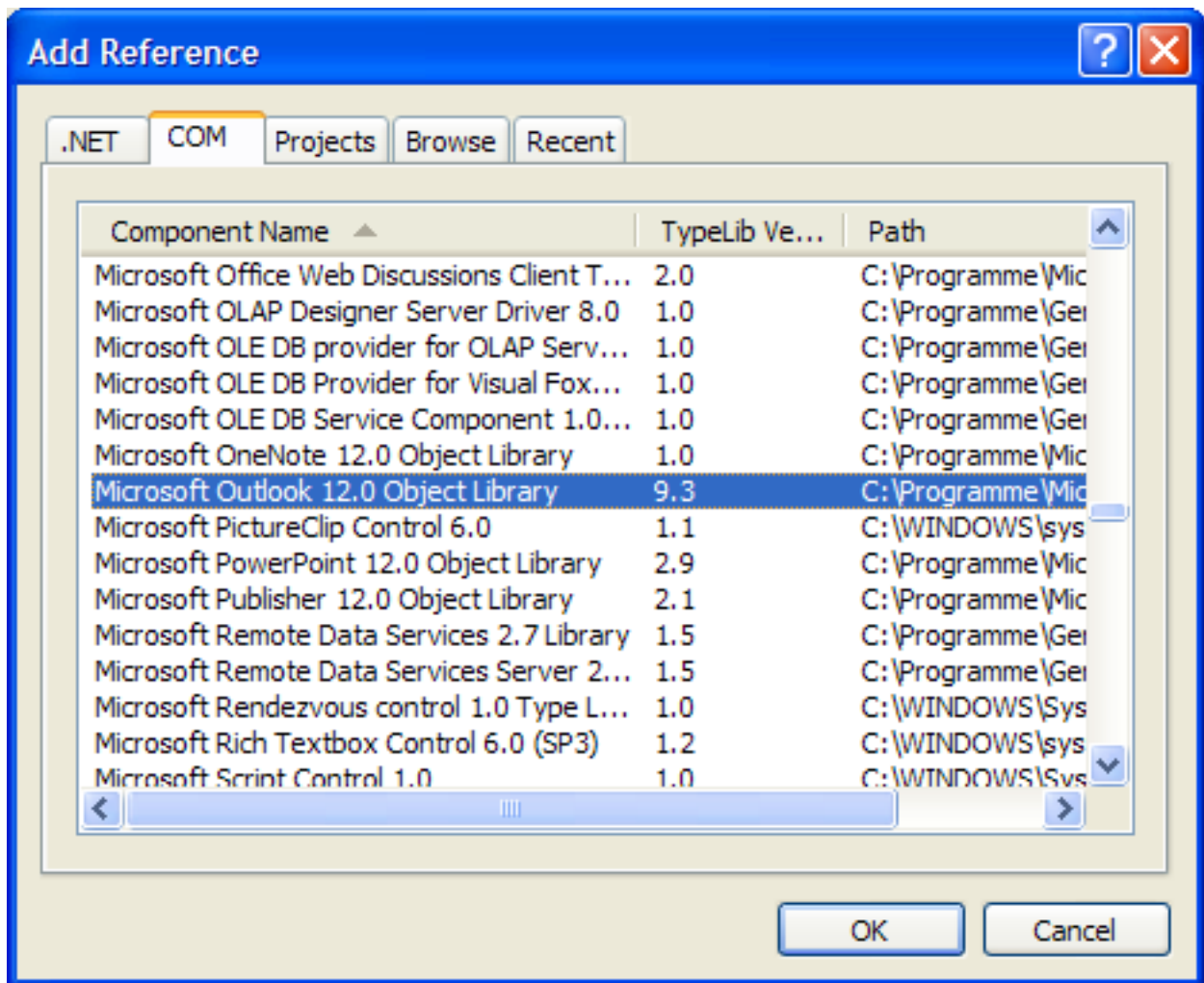
I-B - Créer une solution

Pour démontrer comment cela marche sous le capot, commencez avec une simple application Windows Form qui crée un nouvel email et ajoute un nouveau entête SMTP à celui-ci. Dans cet exemple, vous trouverez un projet disponible en C# et en VB.Net. Ainsi, ouvrez-le avec Visual Studio, choisissez le langage de votre choix et créez une application ressemblant à ceci :



Imaginons que vous soyez un développeur Outlook et vous voulez utiliser le modèle Outlook Object. Vous devez naturellement dans les références de votre projet ajouter les références suivantes:

- Microsoft Office 1X.0 Object Library
- Microsoft Outlook 1X.0 Object Library Note: cela dépend de la version d'Office installée.



Quand vous avez fini avec l'ajout des références, vous pouvez importer les espaces de nom dans votre projet de cette façon:

Code VB.NET

```
Imports Office = Microsoft.Office.Core
Imports Outlook = Microsoft.Office.Interop.Outlook
```

Code C#

```
using Office = Microsoft.Office.Core;
using Outlook = Microsoft.Office.Interop.Outlook;
```

II - Le truc Outlook

Le but est de créer un nouvel email et d'y attacher un nouvel header SMTP. Regardons d'abord comment créer un nouvel email avec le Modèle Outlook Object:

Modèle Outlook Object

```
Private Sub btnSend_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSend.Click

    ' Obtient l'objet Outlook Application
    Dim outlookApplication As Outlook.Application = New Outlook.Application()

    ' Obtient l'objet d'espace de nom
    Dim olNamespace As Outlook.Namespace = _
        outlookApplication.GetNamespace("MAPI")
```

Modèle Outlook Object

```
' Se loggue à la session : ici, nous utilisons un Outlook déjà ouvert
olNamespace.Logon( , , True, True)

' Création d'un nouveau mail et remplissage avec les informations fournies dans le formulaire
Dim newMail As Outlook.MailItem = _
    lookApplication.CreateItem(Outlook.OlItemType.olMailItem)
newMail.To = tbxRecipientAddress.Text
newMail.Subject = tbxSubject.Text
newMail.Body = tbxMessage.Text

newMail.Send()
newMail = Nothing

' logoff de l'espace de nom (session)
olNamespace.Logoff()

' Libère la référence à l'objet Outlook
outlookApplication = Nothing

' Laisse le Garbagecollector faire son boulot

GC.Collect()
GC.WaitForPendingFinalizers()

End Sub
```

La version C++

Version C++

```
private void btnSend_Click(object sender, EventArgs e)
{
    object missing = System.Reflection.Missing.Value;

    // Obtient l'objet Outlook Application
    Outlook.Application outlookApplication = new Outlook.Application();

    // Obtient l'objet d'espace de nom
    Outlook.NameSpace nameSpace = outlookApplication.GetNamespace("MAPI");

    // Se loggue à la session : ici, nous utilisons un Outlook déjà ouvert
    nameSpace.Logon(missing, missing, true, true);

    // Création d'un nouveau mail et remplissage avec les informations fournies dans le formulaire
    Outlook.MailItem newMail = _
        lookApplication.CreateItem(Outlook.OlItemType.olMailItem);
    newMail.To = tbxRecipientAddress.Text;
    newMail.Subject = tbxSubject.Text;
    newMail.Body = tbxMessage.Text;

    newMail.Send();
    newMail = null;

    // logoff de l'espace de nom (session)
    olNamespace.Logoff();

    // Libère la référence à l'objet Outlook
    outlookApplication = Nothing;

    // Laisse le Garbagecollector faire son boulot
    GC.Collect();
    GC.WaitForPendingFinalizers();
}
```

Ce petit snippet devrait simplement envoyer un Email au destinataire que vous aurez renseigné dans le champ **To**:

III - La bibliothèque C++ mixte

Maintenant, vous devez ajouter un nouveau projet à la solution. Choisissez **Fichier > Ajouter > Nouveau projet** et sélectionnez **C++ CLR Class Library**.



J'ai nommé la bibliothèque **MAPIConcubine** parce qu'elle me donner que le Model Outlook Object Model ne me donnera pas. Bien sûr, vous pouvez lui donner le nom de votre choix. Après que vous ayez ajouté la bibliothèque C++ à la solution, ouvrez les propriétés du projet et ajouter mapi32.lib en tant que fichier d'entrée sur le linker.



L'étape suivante est d'ajouter les fichiers d'entête MAPI C++ à votre projet. Ouvrez le fichier **stdafx.h** et ajouter les entêtes suivants dans ce dernier, juste après la déclaration **#pragma once**.

```
#pragma once
#include <MAPIX.h>
#include <MapiUtil.h>
#include <MAPIGuid.h>
#include <MAPITags.h>
```

Compilez ensuite le projet et regardez si cela compile correctement. Si tout se passe bien, vous pouvez continuer. Sinon, vous avez probablement oublié un pré-requis comme le SDK Windows contenant les fichiers d'entête C++ définis dans **Stdafx.h**.

IV - Ajout de fonctionnalités

Maintenant, vous devriez avoir une bibliothèque .Net compilée et liée à la bibliothèque **Extended MAPI**. Allez-y et commencez à ajouter des fonctionnalités à vos projets .Net. Rappelez-vous votre but initial, vous voulez ajouter un entête http à un mail sortant et envoyé via Outlook. Maintenant, vous pouvez continuer et fournir une interface au monde extérieur, qui expose les fonctionnalités désirées.

IV-A - Le fichier d'entête

Les classes C++ ont généralement un fichier d'entête et un fichier de code. Ici, dans votre bibliothèque, le fichier d'entête ressemble à ce qui suit. Vous avez défini une classe .Net, **Fabric**, avec une méthode publique nommée **AddMessageHeader**. La méthode prend trois paramètres : le premier est **MPIObject**, que vous pouvez obtenir depuis les Items Outlook en tant que propriété. Cette propriété fourni l'accès à l'interface **MAPI IUnknown**.

```
#pragma once

using namespace System;
using namespace System::Runtime::InteropServices;

namespace MAPIConcubine
{
    ///
    /// The Fabric Class defines Methods and Functions to access
    /// Extended MAPI Functions from .NET
    ///
    public ref class Fabric
    {
        // private methods and variables
    private:
```

```
// destructor
~Fabric();

///
/// Used to retrieve a human readable ErrorMessage from a
/// MAPI ErrorCode.
///
/// [in]dw - the error number returned from MAPI
/// [returns] Returns the ErrorMessage to the given Errornumber.
String^ GetErrorText(DWORD dw);

// All public visible methods
public:

// construction code
Fabric();

///
/// Adds an InternetHeader to an outgoing Outlook- Email Message
///
/// [in]mapiObject - The MAPIOBJECT property of the Outlook mailItem
/// [in]headerName - The name of the header to add
/// [in]headerValue - The value of the header to add
void AddMessageHeader(Object^ mapiObject, String^ headerName,
String^ headerValue);

};
}
```

Vous exposez seulement les méthodes conformes .Net à l'extérieur. Cela rend facile d'utiliser cette bibliothèque depuis n'importe quel projet .Net. Maintenant vous avez défini l'interface et vous pouvez aller implémenter des fonctionnalités derrière celle-ci. Regardons l'implémentation qui peut être trouvé dans le fichier **MAPIConcubine.cpp**.

L'idée est d'obtenir l'interface **IUnknown** depuis l'objet Outlook, suivi par le **IMessage** et, non pas des moindres, l'interface **IMAPIProp**. Ici, la bibliothèque C++ joue un rôle maître. Vous pouvez maintenant accéder aux fichiers d'entête où toutes les interfaces sont définies et alors compilez à l'aide de bibliothèques externes comme **mapi32.lib** ou toute autre bibliothèque native non managée. Continuez un peu dans le code, là où vous pouvez voir un mix entre les pointeurs traditionnels du C++ et le code non managé (new et *), et le code .Net managé (gcnew and ^). Le piège que vous pouvez voir ici est comment vous passez les variables de chaînes de caractères .Net en LPWSTR non managé, un pointeur vers un *null* Unicode terminant chaque tableau de caractères. .Net vous donne déjà les méthodes correctes dans l'espace de nom **System.Runtime.InteropServices**.

```
/// Ajoute un entête à un mail sortant
///
/// [in]mapiObject - La propriété MAPIOBJECT du mailItem Outlook
/// [in]headerName - Le nom de l'entête à ajouter
/// [in]headerValue - La valeur de l'entête à ajouter
void Fabric::AddMessageHeader(Object^ mapiObject, String^ headerName,
String^ headerValue)
{

// Pointeur vers l'interface IUnknown
IUnknown* pUnknown = 0;

// Pointeur vers IMessage
IMessage* pMessage = 0;

// Pointer to IMAPIProp Interface
IMAPIProp* pMAPIProp = 0;

// Une structure pour la variable MANEDProp
MAPINAMEID* pNamedProp = 0;

// Un tableau de PropertyTags comme résultat pour la méthode GetIDdFromNames.
SPropTagArray* lpTags = 0;
```

```

// Converti les chaines de caractères .Net en blocs de mémoire non-managée
IntPtr pHeaderName = Marshal::StringToHGlobalUni (headerName);
IntPtr pHeaderValue = Marshal::StringToHGlobalUni (headerValue);

// si nous n'avons pas notre MAPIObject rien ne sert de continuer...
if (mapiObject == nullptr)
    throw gcnew System::ArgumentNullException
        ("mapiObject", "The MAPIObject must not be null!");
try
{
    // récupère l'interface IUnknown depuis notre MAPIObject
    // venant d'Outlook.
    pUnknown = (IUnknown*)Marshal::GetIUnknownForObject
        (mapiObject).ToPointer ();

    // essaie de récupérer l'interface IMessage,
    // si on ne l'a pas, autant arrêter de suite.
    if ( pUnknown->QueryInterface
        (IID IMessage, (void*)&pMessage) != S_OK)
        throw gcnew Exception
            ("QueryInterface failed on IUnknown for IID_Message");

    // Essaie de récupérer l'interface IMAPIProp depuis l'interface IMessage,
    // sinon on s'arrête.
    if ( pMessage->QueryInterface
        (IID IMAPIProp, (void*)&pMAPIProp) != S_OK)
        throw gcnew Exception
            ("QueryInterface failed on IMessage for IID_IMAPIProp");

    // vérifie et si on a pas de pointeur, on quitte...
    if (pMAPIProp == 0)
        throw gcnew Exception
            ("Unknown Error on receiving the Pointer to MAPI Properties.");

// Un ID super bien documenté par Google est utilisé pour accéder aux entêtes SMTP d'un message Outlook.
GUID magicId = { 0x00020386, 0x0000, 0x0000,
    { 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46 } };

// Le pointeur vers la structure namedProp
if (MAPIAllocateBuffer(sizeof(MAPINAMEID),
    (LPVOID*)&pNamedProp) != S_OK)
    throw gcnew Exception("Could not allocate memory buffer.");

// L'ID est utilisé pour accéder à la propriété nommée
pNamedProp->lpguid = (LPGUID)&magicId;

// Définit la propriété Nom
pNamedProp->ulKind = MNID_STRING;
pNamedProp->Kind.lpwstrName = (LPWSTR) pHeaderName.ToPointer();

// Appelle GetIDsFromNames pour retrouver la propertyID à utiliser
if (pMAPIProp->GetIDsFromNames
    (1, &pNamedProp, MAPI_CREATE, &lpTags) != S_OK)
    throw gcnew Exception(String::Format
        ("Error retrieving GetIDsFromNames: {0}.", headerName) );

// La variable qui sera passé à la méthode HrSetOneProp.
SPropValue value;
// La macro PROP_TAG créé la valeur correcte depuis la valeur type et la valeur ID
value.ulPropTag = PROP_TAG(PT_UNICODE, PROP_ID
    (lpTags->auiPropTag[0]));
// Ici, nous passons une chaîne unicode à la méthode et définissons la propriété LPSZ
value.Value.LPSZ = (LPWSTR) pHeaderValue.ToPointer();

// La méthode HrSetOneProp est utilisé pour définir la propriété sur l'objet MAPI
if ( HrSetOneProp(pMAPIProp, &value) != S_OK)
    throw gcnew Exception(String::Format
        ("Error setting Header: {0}.", headerName) );
}
catch (Exception^ ex)

```

```

    {
        DWORD dw = GetLastError();
        throw gnew Exception(GetErrorText(dw), ex);
    }
    finally
    {

        if (pNamedProp != 0) MAPIFreeBuffer(pNamedProp);
        if (lpTags != 0) MAPIFreeBuffer(lpTags);

        // libère la mémoire non managée
        Marshal::FreeHGlobal (pHeaderName);
        Marshal::FreeHGlobal (pHeaderValue);

        // Nettoie toutes les références aux objets COM
        if (pMAPIProp!=0) pMAPIProp->Release();
        if (pMessage!=0) pMessage->Release();
        if (pUnknown!=0) pUnknown->Release();
    }
}
    
```

La chose intéressante est comment vous implémentez un bloc **try/catch/finally**. Vous devez faire attention à ne pas obtenir une fuite mémoire, et donc vous devez placer votre code de nettoyage dans la section **finally**. Vous pouvez aussi lancer (**throw**) des exceptions aux classes .Net appelantes. Compilez et vous verrez que vous oubliez quelque chose.

C'est la façon dont vous devez accéder aux ID MAPI définis dans les fichiers d'entête **MAPI**. Il reste juste une action complémentaire à faire maintenant : allez dans votre fichier **Stdafx.h** et incluez les définitions suivantes. Vous devez inclure ces définitions pour chaque ID MAPI que vous utilisez

```

#pragma once

#define INITGUID
#define USES_IID_IMAPIProp
#define USES_IID IMessage

#include <MAPIX.h>
#include <MapiUtil.h>
#include <MAPIGuid.h>
#include <MAPITags.h>
    
```

IV-B - Initialisation et nettoyage

Ahh! Attendez, vous oubliez quelque chose. A chaque fois que vous voulez utiliser quelque chose depuis Mapi Extended, vous devez l'initialiser. Ainsi, vous devez ajouter une routine de construction et une routine de nettoyage à la bibliothèque. Voici, ce à quoi ça ressemble:

```

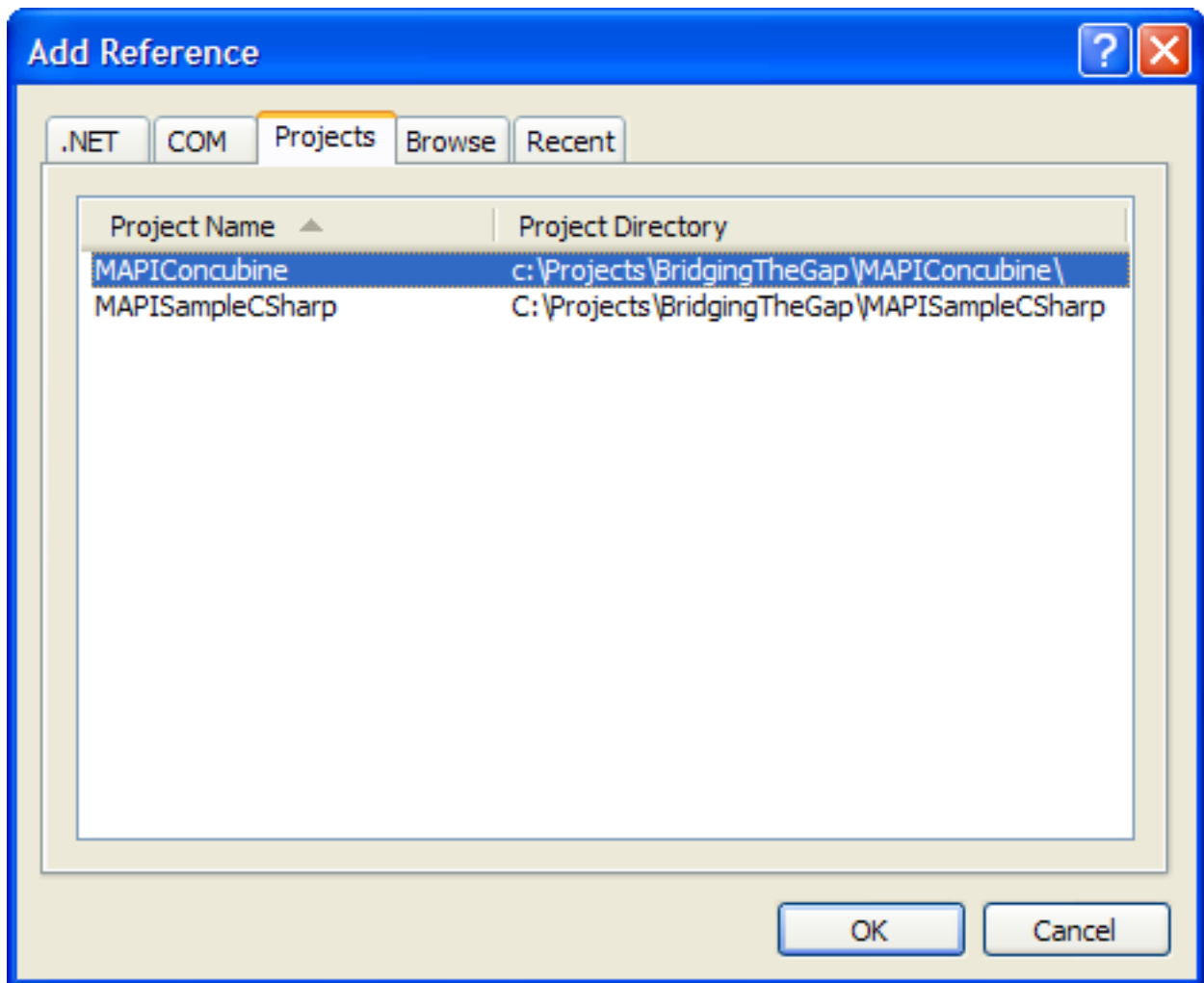
///
/// Code de construction - Nous pouvons initialiser la session MAPI ici
///
Fabric::Fabric ()
{
    MAPIInitialize(0);
}
///
/// Destructeur - Nettoyage de la session ici
///
Fabric::~Fabric ()
{
    MAPIUninitialize();
}
    
```

Egalement, pour une gestion facile des erreurs et du débogage, vous devriez ajouter une méthode qui retourne un message lisible pour l'humain, à l'appelant. Cette méthode pourrait ressembler au code suivant :

```
/// Cette méthode prend un code erreur MAPI en paramètre (GetLastError())  
  
/// et retourne le message d'erreur en chaîne de caractère managée.  
///  
/// [in]dw - le code d'erreur MAPI.  
/// [return] retourne le message d'erreur.  
String^ Fabric::GetErrorText(DWORD dw)  
{  
    LPVOID lpMsgBuf;  
  
    FormatMessage(  
        FORMAT_MESSAGE_ALLOCATE_BUFFER |  
        FORMAT_MESSAGE_FROM_SYSTEM,  
        NULL,  
        dw,  
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),  
        (LPTSTR) &lpMsgBuf,  
        0, NULL );  
  
    String^ result = Marshal::PtrToStringAuto (IntPtr::IntPtr (lpMsgBuf));  
  
    LocalFree(lpMsgBuf);  
  
    return result;  
}
```

V - The usage

Passons aux clients .Net qui pourraient utiliser cette bibliothèque. Ouvrez l'application WinForm - ou simplement VSTO ou un AddIn d'extensibilité - que vous avez créé plus tôt et ajoutez lui une nouvelle référence. Allez dans l'onglet Projets et sélectionné le projet existant : **MAPIConcubine**.



Maintenant, vous pouvez changer le code de votre application Windows Forms et ajoutez lui la nouvelle fonctionnalité. Ca ressemble simplement à ceci:

```
' Créez un nouveau mail et remplissez le avec les informations fournies dans le formulaire.
Dim newMail As Outlook.MailItem = _
    outlookApplication.CreateItem(Outlook.OlItemType.olMailItem)
' Ici, nous pouvons utiliser notre bibliothèque C++
Dim fabric As MAPIConcubine.Fabric = New MAPIConcubine.Fabric()
' Nous passons la propriété MAPIObject notre bibliothèque et nos paramètres
' notez qu'en VB.Net, vous ne pouvez pas voir la propriété MAPIOBJECT
' mais soyez sûrs qu'elle est bien là
fabric.AddMessageHeader(newMail.MAPIOBJECT, tbxHeaderName.Text, _
    tbxHeaderValue.Text)
' C'est tout
fabric = Nothing
newMail.To = tbxRecipientAddress.Text
newMail.Subject = tbxSubject.Text
newMail.Body = tbxMessage.Text
' Envoi de l'email
newMail.Send()
```

Version C++

```
// Crée un nouveau mail et le remplit avec les informations fournies dans le formulaire
Outlook.MailItem newMail = outlookApplication.CreateItem
(Outlook.OlItemType.olMailItem) as Outlook.MailItem;
// Ici, nous pouvons utiliser notre bibliothèque C++
MAPIConcubine.Fabric fabric = new MAPIConcubine.Fabric();
// Nous passons la propriété MAPIObject notre bibliothèque et nos paramètres
```

```
// notez qu'en VB.Net, vous ne pouvez pas voir la propriété MAPIOBJECT
// mais soyez sûrs qu'elle est bien là
fabric.AddMessageHeader(newMail.MAPIOBJECT, tbxHeaderName.Text,
    tbxHeaderValue.Text);
// c'est tout
fabric = null;
newMail.To = tbxRecipientAddress.Text;
newMail.Subject = tbxSubject.Text;
newMail.Body = tbxMessage.Text;
newMail.Send();
```

C'est tout. Le reste est très facile. Vous pouvez vérifier les entêtes du mail dans la boîte de réception et vous y verrez vos propres entêtes. Dans Outlook, vous pouvez les en ouvrant l'email et en affichant le menu des Options.

VI - Lire l'entête

Dans ce chapitre, vous aller apprendre comment retrouver les entêtes sauvegardés et comment retrouver tous les entêtes de message de transport. La théorie est simple : vous utilisez le GUID pour accéder aux entêtes Internet et la méthode **GetIDSFromNames**. Cela vous permet de retrouver le bon **propTagId** que vous utilisez dans la méthode **HrGetOneProp**. Si vous êtes dans un environnement Exchange, vous pouvez lire les entêtes d'email personnalisé avec le code suivant : Notez que pour une lecture aisée, le bloc **catch/finally** a été supprimé. Comme plus haut, référez vous aux fichiers sources pour plus de détails.

```
///
/// Lis l'entête depuis un message Outlook
///
/// [in] mapiObject - la propriété MAPIOBJECT de l'objet mailItem
/// [in] headerName - Nom de l'entête à retrouver.
/// [returns] headerValue - la valeur de l'entête avec le nom donné
String^ Fabric::ReadMessageHeader(Object^ mapiObject, String^ headerName)
{
    // Pointeur vers l'interface IUnknown
    IUnknown* pUnknown = 0;
    // Pointeur vers l'interface IMessage
    IMessage* pMessage = 0;
    // Pointeur vers l'interface IMAPIProp
    IMAPIProp* pMAPIProp = 0;
    // La structure pour la variable MANEDProp
    MAPINAMEID* pNamedProp = 0;
    // Un tableau de PropertyTags comme résultat pour la méthode GetIDdFromNames Method.
    SPropTagArray* lpTags = 0;
    // Pointeur vers la structure qui reçoit le résultat de HrGetOneProp
    LPSPropValue lpSPropValue = 0;
    // Converti les chaînes de caractères .Net en blocs de mémoire non managé
    IntPtr pHeaderName = Marshal::StringToHGlobalUni (headerName);
    // Si nous n'avons pas notre MAPIObject, autant tout arrêter
    if (mapiObject == nullptr)
        throw gcnew System::ArgumentNullException ("mapiObject", "The MAPIObject must not be null!");
    try
    {
        // Récupère l'interface IUnknon depuis notre MAPIObject venant d'Outlook
        pUnknown = (IUnknown*)Marshal::GetIUnknownForObject(mapiObject).ToPointer ();
        // Essaie de récupérer l'interface IMessag, si on ne peut pas, on arrête
        if ( pUnknown->QueryInterface (IID IMessage, (void*)&pMessage) != S_OK)
            throw gcnew Exception("QueryInterface failed on IUnknown for IID_Message");
        // essaie de récupérer l'interface depuis l'interface IMessage, sinon on arrête.
        if ( pMessage->QueryInterface (IID IMAPIProp, (void*)&pMAPIProp) != S_OK)
            throw gcnew Exception("QueryInterface failed on IMessage for IID_IMAPIProp");
        // double vérification
        if (pMAPIProp == 0)
            throw gcnew Exception("Unknown Error on receiving the Pointeur versMAPI Properties.");

        // Un ID trouvable via google et « bien documenté » pour accéder aux entêtes SMTP d'un message Outlook
        GUID magicId =
        {
            x00020386, 0x0000, 0x0000,
```

```
        {
            0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46
        }
    };
    // Le pointeur vers la structure namedProp
    if (MAPIAllocateBuffer(sizeof(MAPINAMEID), (LPVOID*)&pNamedProp) != S_OK)
        throw gcnw Exception("Could not allocate memory buffer.");
    // L'ID utilisé pour accéder la propriété nommée
    pNamedProp->lpguid = (LPGUID)&magicId;
    // Définit PropertyName
    pNamedProp->ulKind = MNID_STRING;
    NamedProp->Kind.lpwstrName = (LPWSTR) pHeaderName.ToPointer();
    // Appelle GetIDsFromNames pour retrouver propertyID à utiliser
    if (pMAPIProp->GetIDsFromNames(1, &pNamedProp, STGM_READ, &lpTags) != S_OK)
        throw gcnw Exception(String::Format ("Error retrieving GetIDsFromNames: {0}.", headerName));

    // La macro PROP_TAG créé la bonne valeur pour Type PropertyID
    ULONG propTag = PROP_TAG(PT_UNICODE, PROP_ID(lpTags->aulPropTag[0]));
    // utilise la méthode HrGetOneProp pour retrouver la propriété.
    // Le pointeur lpPropValue pointe vers la valeur de résultat
    if (HrGetOneProp(pMAPIProp, propTag, &lpSPropValue) != S_OK)
        throw gcnw Exception("HrGetOneProp failed for named property !");
    // créé une chaîne de caractère managée depuis la chaîne non managée
    return gcnw String( lpSPropValue->Value.lpszW );
}
catch (Exception^ ex)    {        ...    }
finally    {        ...    }
}
```

L'utilisation est identique à l'autre méthode. Créer un objet fabriqué et passer **mailitem.MAPIOBJECT** à la méthode ainsi que le nom de l'entête personnalisé.

VII - Lire le nom du profil Outlook courant.

```
private void button1_Click(object sender, EventArgs e)
```

```
{
```

```
    // Create an Outlook session
```

```
    obj Form1
```

```
    str
```

```
    str
```

AnotherProfile

Test

```
    //
```

```
    Out
```

```
    //
```

```
    Out
```

```
    //
```

```
    olN
```

```
    // Use the fabric to get the current profilename
```

```
    MAPIConcubine.Fabric fabric = new MAPIConcubine.Fabric();
```

```
    try
```

```
    {
```

```
        string currentProfileName = fabric.GetProfileName(olNamespace.MAPIOBJECT);
```

```
        MessageBox.Show(this, String.Format("The current profilename was: {0}", curre
```

```
    }
}
```

```
catch (System.Exception ex)
```

Dans cette leçon, vous apprendrez comment passer outre un problème courant pour les développeurs Outlook. Vous apprendrez comment lire le nom du profil de la session MAPI courante. Dans le modèle Outlook Object, il n'y a aucun moyen de déterminer le nom du profil que l'utilisateur a démarré. Avec le snippet suivant, vous pouvez résoudre ce problème. L'idée est d'obtenir l'objet session et d'utiliser la méthode **OpenProfileSection** pour retrouver les informations du profil Outlook courant. Notez que pour une lecture plus aisée, le bloc **catch/finally** a été enlevé. Comme plus haut, référez vous aux fichiers sources pour plus de détails

```
///
/// Retourne le nom du profil MAPI de la session courante
///
/// [in] mapiObject - la MAPIOBJECT de Outlook Application.Session
/// [returns] - le nom du profil actuellement utilisé
String^ Fabric::GetProfileName(Object^ mapiObject)
{
    // Le résultat retourné à la méthode appelante
    String^ result = nullptr;

    // Pointeur vers l'interface IUnknown
    IUnknown* pUnknown = 0;

    // pointeur vers l'interface de la session MAPI
```

```

LMAPISESSION lpMAPISession = 0;

// pointeur vers l'interface profilesection
LPPROFSECT lpProfileSection = 0;

// pointeur vers une structure qui reçoit le résultat depuis HrGetOneProp
LPSPROPVALUE lpSPROPVALUE = 0;

if (mapiObject == nullptr)
    throw gcnew System::ArgumentNullException ("mapiObject",
        "The MAPIObject must not be null!");

try
{
    // récupère l'interface IUnknown depuis notre MAPIObject
    pUnknown =
        (IUnknown*)Marshal::GetIUnknownForObject(mapiObject).ToPointer ();

    if ( pUnknown->QueryInterface (IID_IMAPISESSION,
        (void**)&lpMAPISession) != S_OK)
        throw gcnew Exception(
            "QueryInterface failed on IUnknown for IID_IMAPISESSION");

    // Utilise OpenProfileSection de l'objet MAPISESSION
    // pour récupérer un pointeur vers profilesection courante
    if( lpMAPISession->OpenProfileSection(
        (LMAPIUID)GLOBAL_PROFILE_SECTION_MAPIUID,
        NULL,STGM_READ, &lpProfileSection) != S_OK)
        throw gcnew Exception("OpenProfileSection method failed!");

    // utilise la méthode HrGetOneProp pour retrouver la propriété
    // le pointeur lpPROPVALUE pointe vers la valeur résultat
    if (HrGetOneProp(lpProfileSection,
        PR_PROFILE_NAME_W,&lpSPROPVALUE) != S_OK)
        throw gcnew Exception(
            "HrGetOneProp failed for property PR_PROFILE_NAME_W !");

    // créé une chaîne managée depuis la chaîne non managée.
    return gcnew String( lpSPROPVALUE->Value.lpszW );

}
catch (Exception^ ex)
{
    ...
}
finally
{
    ...
}
}
    
```

Maintenant, c'est facile d'obtenir l'information reliée au profil sur lequel l'utilisateur est logué. Juste pour référence, vous implémentez une méthode qui énumère tous les profils MAPI disponibles pour l'utilisateur courant. Cette information est stockée dans le registre et ainsi vous ouvrez simplement la clé '**HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles**' et énumérer toutes les sous-clés (une pour chaque profil).

```

///
/// Retourne les profilename MAPI disponibles.
///
/// [returns] - un tableau de string de tous les profilename disponibles.
array<string^,1 />^ Fabric::EnumProfileNames()
{
    String^ registryPath =
        "Software\Microsoft\Windows NT\CurrentVersion" +
        "\\Windows Messaging Subsystem\Profiles";
    RegistryKey^ key = nullptr;
    try
    
```

```
{
    key = Registry::CurrentUser->OpenSubKey ( registryPath );
    return key->GetSubKeyNames ();
}
catch(System::Exception^ ex)
{
    throw ex;
}
finally
{
    key->Close();
}
```

L'application de test fourni dans les sources démontre comment utiliser cette méthode. Passez simplement la propriété **namespace.MAPIOBJECT**.

Voici le code C#:

```
// Créé une session Outlook
object missing = System.Reflection.Missing.Value;
// récupère l'objet Outlook
Application Outlook.Application outlookApplication = new Outlook.Application();
// récupère l'objet d'espace de nom
Outlook.NameSpace olNamespace = outlookApplication.GetNamespace("MAPI");
// Se logue à la session. Ici, nous utilisons la combobox pour se connecter à un session particulière
// Si c'est déjà une instance Outlook avec une session, alors ce profil est utilisé
olNamespace.Logon(comboBox1.Text , missing, true, true);
// Utilise la fabrique pour obtenir le nom de profile courant
MAPIConcubine.Fabric fabric = new MAPIConcubine.Fabric();
try
{
    // transmet le namespace.MAPIOBJECT pour récupérer le profilename
    string currentProfileName = fabric.GetProfileName(olNamespace.MAPIOBJECT);
    MessageBox.Show(this, String.Format("The current profilename was: {0}", currentProfileName));
}
catch (System.Exception ex) {
    MessageBox.Show(this, ex.Message);
}
finally {
    fabric = null;
    olNamespace.Logoff();
    olNamespace = null;
    outlookApplication = null;
    GC.Collect();
    GC.WaitForPendingFinalizers();
}
```

VIII - Définit la couleur du label de rendez-vous

The screenshot shows the Outlook 2007 interface for creating an appointment. The ribbon includes 'Termin', 'Einfügen', 'Text formatieren', and 'Add-Ins'. The 'Termin' ribbon has buttons for 'Speichern & schließen', 'Teilnehmer einladen', 'Terminplanung', 'Optionen', 'Rechtschreibung', and 'Besprechungsnotizen'. The appointment form shows 'Betreff: Testtermin' and 'Ort:'. The start time is 'Mi 27.06.2007 20:00'. A custom form 'Form1' is overlaid, showing a dropdown menu with 'Birthday' selected and 'Set' and 'Get' buttons. A message box is displayed in the foreground, stating 'The current AppointmentLabel Color is : Birthday'.

Dans cette section, vous allez apprendre comment définir et obtenir la couleur d'un label dans un Item de Rendez-vous Outlook (Appointment). Avec les versions d'Outlook 2002 et postérieur, vous avez la possibilité de colorier vos rendez-vous. Regardez ici pour plus d'information. Dans le passé, vous deviez utiliser CDO ou l'une des bibliothèques externes mentionnées plus haut. Ici se trouve le code source qui démontre comment utiliser Extended MAPI pour réaliser la même chose depuis .Net. Dans le fichier d'entête, vous définissez les couleurs disponibles pour les labels de rendez-vous :

```

/// Defines the available Colors for AppointmentLabels

enum class AppointmentLabelColor
{
    None = 0,           // Default Color
    Important = 1,     // Red
}

```

```
Business = 2, // Blue
Personal = 3, // Green
Vacation = 4, // Grey
Deadline = 5, // Orange
Travel_Required = 6, // Light-Blue
Needs_Preparation = 7, // Grey
Birthday = 8, // violett
Anniversary = 9, // turquoise
Phone_Call = 10 // Yellow
} ;
```

Quand vous avez fini avec le fichier d'entête, continuez et implémentez la méthode **SetAppointmentLabelColor** comme ci-dessous. Notez que le bloc **catch/finally** a été enlevé pour une lecture plus aisée

```
///
/// Sets the AppointmentLabelColor for a Outlook- Appointment
///
/// [in] mapiObject - the MAPIOBJECT property of
/// the outlook appointmentItem object.
/// [in] the desired colortype that you wish to set

void Fabric::SetAppointmentLabelColor(Object^ mapiObject,
AppointmentLabelColor color)
{
    // Pointeur vers l'interface IUnknown
    IUnknown* pUnknown = 0;

    // Pointeur vers l'interface IMessage
    IMessage* pMessage = 0;

    // Pointeur vers l'interface IMAPIProp
    IMAPIProp* pMAPIProp = 0;

    // Une structure pour la variable NAMEDProp
    MAPINAMEID* pNamedProp = 0;

    // Un tableau de PropertyTags comme résultat pour la méthode GetIDdFromNames.
    SPropTagArray* lpTags = 0;

    if (mapiObject == nullptr) throw gcnew System::ArgumentNullException (
        "mapiObject", "The MAPIOBJECT must not be null!");
    try
    {
        pUnknown =
            (IUnknown*)Marshal::GetIUnknownForObject(mapiObject).ToPointer ();

        if ( pUnknown->QueryInterface (IID IMessage,
            (void*)&pMessage) != S_OK)
            throw gcnew Exception(
                "QueryInterface failed on IUnknown for IID_Message");

        if ( pMessage->QueryInterface (IID IMAPIProp,
            (void*)&pMAPIProp) != S_OK)
            throw gcnew Exception(
                "QueryInterface failed on IMessage for IID_IMAPIProp");

        // double check
        if (pMAPIProp == 0)
            throw gcnew Exception(
                "Unknown Error on receiving the Pointer to MAPI Properties.");

        // ID trouvé via google.
        GUID magicId =
        {
            0x00062002, 0x0000, 0x0000,
            {
                0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46
            }
        }
    }
}
```

```

    }
};
LONG propertyName = 0x8214;

// Le pointeur vers la structure namedProp
if (MAPIAllocateBuffer(sizeof(MAPINAMEID),
    (LPVOID*)&pNamedProp) != S_OK)
    throw gcnew Exception("Could not allocate memory buffer.");

// L'ID est utilisé pour accéder à la propriété
pNamedProp->lpguid = (LPGUID)&magicId;

// Définir PropertyName
pNamedProp->ulKind = MNID_ID;
pNamedProp->Kind.lID = propertyName;

if(pMAPIProp->GetIDsFromNames(1, &pNamedProp,
    MAPI_CREATE, &lpTags) != S_OK)
    throw gcnew Exception(String::Format (
        "Error retrieving GetIDsFromNames: {0}.",propertyName) );

SPropValue value;

value.ulPropTag = PROP_TAG(PT_LONG, PROP_ID(lpTags->aulPropTag[0]));
value.Value.l = (LONG)color;

if( HrSetOneProp(pMAPIProp, &value) != S_OK)
    throw gcnew Exception(String::Format (
        "Error setting AppointmentLabelColor: {0}.",color) );

// Sauvegarde les changements sur l'item
pMessage->SaveChanges (KEEP_OPEN_READWRITE);
}
catch (Exception^ ex)
{
    ...
}
finally
{
    ...
}
}

```

La méthode pour récupérer le label est implémentée comme suit :

```

GUID magicId =
{
    0x00062002, 0x0000, 0x0000,
    {
        0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46
    }
};
LONG propertyName = 0x8214;

// Le pointeur vers la structure namedProp
if (MAPIAllocateBuffer(sizeof(MAPINAMEID), (LPVOID*)&pNamedProp) != S_OK)
    throw gcnew Exception("Could not allocate memory buffer.");

pNamedProp->lpguid = (LPGUID)&magicId;

pNamedProp->ulKind = MNID_ID;
pNamedProp->Kind.lID = propertyName;

if(pMAPIProp->GetIDsFromNames(1, &pNamedProp, STGM_READ, &lpTags) != S_OK)
    throw gcnew Exception(String::Format (
        "Error retrieving GetIDsFromNames: {0}.",propertyName) );

// La macro PROP_TAG créé la bonne valeur pour Type et PropertyID
ULONG propTag = PROP_TAG(PT_LONG, PROP_ID(lpTags->aulPropTag[0]));

```

```
// utilise la méthode HrGetOneProp pour retrouver la propriété
// Le pointeur lpPropValue vers la valeur résultant
if (HrGetOneProp(pMAPIProp,propTag,&lpSPropValue) != S_OK)
    throw gcnw Exception("HrGetOneProp failed for named property !");

if (( lpSPropValue->Value.l > 0) && ( lpSPropValue->Value.l < 11))
    return (Fabric::AppointmentLabelColor ) lpSPropValue->Value.l;
else
    return Fabric::AppointmentLabelColor::Default;
```

IX - Notes

Lorsque vous déployez cette DLL avec votre application:

- La DLL devrait être compilée en mode release
- Vous devriez redistribuer les bibliothèques du CLR 8.0 avec (Pré-requis dans le package d'installation MSI)
- VSTO: pour chaque DLL qui est utilisée depuis votre add-in, vous devez régler les stratégies de sécurité (action personnalisée du package d'installation MSI)