```
void Form1::SetMsgInTxtBox(String^ szMsg){

  if(textBox1->InvokeRequired)
    textBox1->Invoke(m_MsgTxtBoxDlgt, szMsg);
  else
    SetMsgInTxtBoxIhm(szMsg);
}

void Form1::SetMsgInTxtBoxIhm(String^ szMsg){

  textBox1->Text += (szMsg + "\r\n");
}
```

Voici la fonction qui sera appelée lorsque notre singleton déclenchera un événement. Ici par contre nous devons gérer l'aspect crossthread. Les contrôles du framework possède le membre InvokeRequired. Si ce membre contient la valeur true, alors cela signifie que nous sommes dans un thread différent du thread du contrôle en question : if(textBox1->InvokeRequired) . Si c'est le cas, alors nous utilisons la méthode Invoke présent pour tous les contrôles. Nous passons la référence sur la fonction à exécuter ainsi que l'argument de type String. L'appel à Invoke va déclencher la fonction suivante : SetMsgInTxtBoxIhm , et cette fois ci nous serons dans le même thread que le contrôle, et nous pourrons mettre à jour le label sans déclenchement d'exception.

Nous aurions pu utiliser cette construction suivante :

```
void Form1::SetMsgInTxtBox(String^ szMsg){

  if(textBox1->InvokeRequired)
    textBox1->Invoke(gcnew MessageDelegate(this, &Form1::SetMsgInTxtBox), szMsg);
  else
    textBox1->Text += (szMsg + "\r\n");
}
```

Ici nous ne faisons que déclencher la même méthode une deuxième fois. Mais au deuxième appel faisant suite à Invoke, nous serons dans le thread correct du contrôle, et le else sera effectué. Cette construction nous épargne d'une fonction et d'une référence. C'est plus clair et plus léger mais... il y a un mais.

Première chose, à chaque appel à Invoke, un objet est créé dans le Garbage Collector (gcnew). Certes, le Framework gère très bien ses objets, mais si vous considérez que le serveur risque de recevoir de multiples données, imaginons plusieurs à la seconde, conservez une référence (m_MsgTxtBoxDlgt) soulagera le GC. Donnons-lui le moins de travail lorsque cela est possible et cohérent.

Deuxième chose. Ecrire une fonction dédiée (void Form1::SetMsgInTxtBoxIhm(String^ szMsg)), permettra d'appeler celle-ci directement depuis l'IHM. Dans le cas d'un appel avec une boucle par exemple, on s'épargnera le test à InvokeRequired.

Ces considérations discutées, vous serez à même de faire le choix qui correspondra à l'architecture et aux exigences de votre application.

I-I Description des fonctions dans Form1_ctrlevent.cpp

```
//-----------------------------------------------------------------------------
// Form1_ctrlevent.cpp
//-----------------------------------------------------------------------------
#include "stdafx.h"
#include "Form1.h"

using namespace server;

System::Void Form1::button1_Click(System::Object^ sender, System::EventArgs^ e){

  if(!m_cAppServer->InitServer())
    label2->Text = "Impossible de lancer le serveur !";
  else
    label1->Text = "STATUS : SERVER ACTIF";
}

System::Void Form1::button2_Click(System::Object^ sender, System::EventArgs^ e){

  if(!m_cAppServer->StopServer())
    label2->Text = "Impossible d'arrêter le serveur !";
  else
    label1->Text = "STATUS : SERVER NON ACTIF";
}
```

Ici rien de bien compliqué. Lors d'un clic sur le bouton 1, nous initialisons le serveur et lors d'un clic sur le bouton 2 nous stoppons le serveur. En fonction de la valeur de retour, nous mettons à jour les labels de l'interface graphique.

I-J Le fichier stdafx.h

```
// stdafx.h : fichier Include pour les fichiers Include système standard,
// ou les fichiers Include spécifiques aux projets qui sont utilisés fréquemment,
// et sont rarement modifiés
#pragma once

// TODO : faites référence ici aux en-têtes supplémentaires nécessaires au programme
using namespace System;
using namespace System::Windows::Forms;
using namespace System::Net;
using namespace System::Net::Sockets;
using namespace System::Text;
using namespace System::Threading;

#include <msclr\lock.h>

#include "cformevent.h"
#include "cserver.h"
#include "cappserver.h"
#include "..\common\definition.h"
```

Comme je suis un développeur fainéant, je mets beaucoup de définition dans ce fichier. Ca accélère les compilations et ça évite d'écrire les chemins de fichiers dans tous les .h ou .cpp. Ca permet aussi de voir rapidement quels sont les namespace et les fichiers nécessaires à notre programme. La règle c'est de mettre les chemins de fichier qui ne sont pas modifiés ou très peu.

Le fichier « definition.h » est rangé dans un répertoire common accessible aux deux projets de la solution. C'est une manière d'éviter d'avoir deux versions de ce fichier, et si vous le modifiez les deux projets resteront à jour. A condition de recompiler bien sûr.