Notre gestionnaire d'exception est identique aux fonctions précédentes, la différence c'est que nous nous assurons que la référence à notre classe est valide : if(Server) …

Le bloc finally sera lui toujours exécuté une fois les traitements terminés. Nous fermons proprement la connexion sous-jacente, c'est une manière de signaler au client que nous arrêtons la communication. Nous vérifions avant que les références sont bien valides.

```
void CServer::OnServerReport(ServerReportEventArgs^ e){

  m_SyncCtx->Post(gcnew SendOrPostCallback(this, &CServer::ServerReport), e);
}

void CServer::ServerReport(Object^ e){

  OnServerReportEvent(this, (ServerReportEventArgs^)e);
}
```

Dans la fonction OnServerReport, nous passons en paramètre notre argument personnalisé. Ici nous utilisons notre contexte de synchronisation afin de faire remonter l'évènement dans le thread courant de l'IHM. Nous utilisons la méthode Post qui prends en paramètre une référence sur une fonction callback, qui sera la fonction ServerReport, juste en dessous. Le deuxième paramètre est l'argument personnalisé.

Lorsque nous serons dans la fonction ServerReport, nous nous trouverons dans le même thread que celui de l'interface graphique. Nous pouvons alors déclencher l'évènement à proprement parler : OnServerReportEvent(this, (ServerReportEventArgs^)e); .

I-G Description des fonctions dans cappserver.cpp

```
//-------------------------------------------------------------------------------------
// cappserver.cpp
//-------------------------------------------------------------------------------------
#include "stdafx.h"

void CAppServer::ShellNetMessage(NetworkStream^ stream, IPAddress^ ip){

  array<Byte>^ btMsg = gcnew array<Byte>(4);
  int iBytes = stream->Read(btMsg, 0, btMsg->Length);

  if(iBytes == 4){

    iBytes = BitConverter::ToInt32(btMsg, 0);

    switch(iBytes){

      case MSG_ID_PING:

        {
          array<Byte>^ btMsg = gcnew array<Byte>(9);
          btMsg = Encoding::Default->GetBytes("SERVER OK");

          stream->Write(btMsg, 0, btMsg->Length);
          OnServerReport(gcnew ServerReportEventArgs("Le server a reçu un ping : " + ip->ToString()));
        }
        break;

      case MSG_ID_SENDMSG:

        {
          array<Byte>^ btMsg = gcnew array<Byte>(1024);
          Int32 bytes = stream->Read(btMsg, 0, btMsg->Length);

          CFormEvent::GetCFormEvent()->MsgIhm(Encoding::Default->GetString(btMsg, 0, bytes));

          btMsg = Encoding::Default->GetBytes("MESSAGE RECU");

          stream->Write(btMsg, 0, 12);
          OnServerReport(gcnew ServerReportEventArgs("Le server a reçu un message : " + ip->ToString()));
        }
        break;

      default:
        OnServerReport(gcnew ServerReportEventArgs("ShellNetMessage  valeur inconnue : " + iBytes.ToString()));
    }
  }
}
```

Au début de la méthode, nous initialisons un tableau de 4 Byte. Nous lisons les données de la connexion : stream->Read(btMsg, 0, btMsg->Length); . Nous passons en paramètre notre tableau, le point d'origine du tableau et la taille du tableau. Nous vériffions ensuite que nous avons bien lu 4 Byte.

Ensuite la valeur du tableau est convertie en un entier : iBytes = BitConverter::ToInt32(btMsg, 0); .

Ici deux cas possibles. L'entier vaut 0 (MSG_ID_PING) ou 1 (MSG_ID_SENDMSG). Ces deux valeurs sont définies dans le fichier “defnition,h”, qui sera commun aux deux applications.

```
//-------------------------------------------
// definition.h
//-------------------------------------------
#ifndef DEFINITION_H
#define DEFINITION_H

#define SERVER_PORT   8888
#define SERVER_IP     "127.0.0.1"

#define MSG_ID_PING       0
#define MSG_ID_SENDMSG    1

#endif
```

Nous définissons aussi la valeur du port d'écoute du serveur et son adresse IP.

Le serveur peut donc recevoir deux types de message que nous venons de définir. La valeur MSG_ID_PING est utilisé pour savoir depuis le client si le serveur est bien en écoute. La valeur MSG_ID_SENDMSG indiquera au serveur que le client lui transmet un message texte.

Pour le premier cas MSG_ID_PING, nous ne faisons que répondre au client en lui transmettant une chaîne de caractère. Nous initialisons un tableau de Byte : array<Byte>^ btMsg = gcnew array<Byte>(9); , d'une taille fixée à 9 et nous remplissons celui-ci : btMsg = Encoding::Default->GetBytes("SERVER OK"); . La classe du Framework Encoding est définie dans le namespace System::Text; .

Nous écrivons ensuite ces données sur le flux réseau : stream->Write(btMsg, 0, btMsg->Length); . L'utilisation des paramètres est équivalent à la fonction de lecture.

Enfin, nous signalons l'évènement (OnServerReport), en indiquant l'adresse IP du client (ip->ToString()). Nous quittons le switch lors du break.

Pour le deuxième cas MSG_ID_SENDMSG, nous récupérons la chaîne de caractère transmise par le client. Nous commençons par initialiser un tableau de 1024 Byte : array<Byte>^ btMsg = gcnew array<Byte>(1024); . Cette initialisation remplit tous les champs du tableau avec la valeur zéro.

Puis nous lisons les données de la connexion : Int32 bytes = stream->Read(btMsg, 0, btMsg->Length); . Bien sûr, côté client, nous allons nous assurer que celui-ci ne pourra pas transmettre plus de 1024 Byte. La valeur Int32 bytes, va récupérer le nombre de Byte lu sur la connexion. Celui nous servira pour la conversion en String.

Ensuite nous transmettons la chaîne récupérée à notre interface graphique. Cette fois-ci nous n'allons pas utiliser le gestionnaire d'évènements de la classe CServer, mais le gestionnaire de la classe CFormEvent : CFormEvent::GetCFormEvent()->MsgIhm(Encoding::Default->GetString(btMsg, 0, bytes)); . C'est la où le singleton prends tout son sens. La construction CformEvent::GetCFormEvent() permet de récupérer une référence sur notre singleton puis nous appelons la méthode MsgIhm qui déclenchera l'évènement. Je rappelle que la classe CAppServer n'a aucune référence sur la classe de l'IHM. Notre singleton permet ce lien. Le paramètre de MsgIhm est une String et nous utilisons la méthode GetString du namespace Encoding. Le dernier paramètre bytes de la méthode GetString est la longueur de la chaîne transmise par le client. Valeur que nous avons récupérée lors de la lecture du flux.

Nous signalons alors au client que nous avons bien reçu le message. C'est la même façon de procéder que lors de la réception du MSG_ID_PING. On copie la chaîne "MESSAGE RECU" dans notre tableau de Byte et nous écrivons cette valeur sur la connexion. Enfin nous signalons au serveur qui a transmis ce message par la méthode habituelle : OnServerReport(gcnew ServerReportEventArgs("Le server a reçu un message : " + ip->ToString())); .

Le switch comporte une case default, à partir de laquelle nous indiquons au serveur que nous venons de recevoir une valeur que celui-ci ne gère pas. Cela peut-être très pratique lorsque l'on se trouve en phase de débogage.

I-H Description des fonctions dans Form1_misc.cpp

```
//-----------------------------------------------------------------------------
// Form1_misc.cpp
//-----------------------------------------------------------------------------
#include "stdafx.h"
#include "Form1.h"

using namespace server;

void Form1::Init(){

  m_cAppServer = gcnew CAppServer(SERVER_PORT);

  m_cAppServer->OnServerReportEvent += gcnew CAppServer::ServerReportHandler(this, &Form1::ServerReport);

  CFormEvent::GetCFormEvent()->OnMsgIhmDlgt += gcnew MessageDelegate(this, &Form1::SetMsgInTxtBox);

  m_MsgTxtBoxDlgt = gcnew MessageDelegate(this, &Form1::SetMsgInTxtBoxIhm);
}
```

Dans cette fonction d'initialisation nous créons un objet serveur avec comme paramètre la valeur du port qui est définie dans le fichier “definition,h”.

Puis nous nous abonnons au gestionnaire d'évènement du serveur. Nous lui passons en paramètre une référence sur notre form et l'adresse d'une fonction à appeler lorsqu'un événement sera déclenché (&Form1::ServerReport).

Nous faisons de même avec le gestionnaire d'évènement du singleton. Les paramètres sont identiques, seul la signature des délégués diffère.

Enfin nous initialisons m_MsgTxtBoxDlgt. Il référencera en permanence l'adresse de notre fonction SetMsgInTxtBoxIhm. Nous verrons un peu plus loin l'avantage à conserver une référence sur cette fonction.

```
void Form1::ServerReport(Object^ sender, CAppServer::ServerReportEventArgs^ e){

   label2->Text = e->MESSAGE;
}
```

Voici la fonction qui sera appelée lorsque le serveur déclenchera un événement. Nous mettons simplement à jour le label de notre form. Ici Pas de gestion crossthread, le serveur s'en ait chargé.