ANNEXE CODES SOURCES

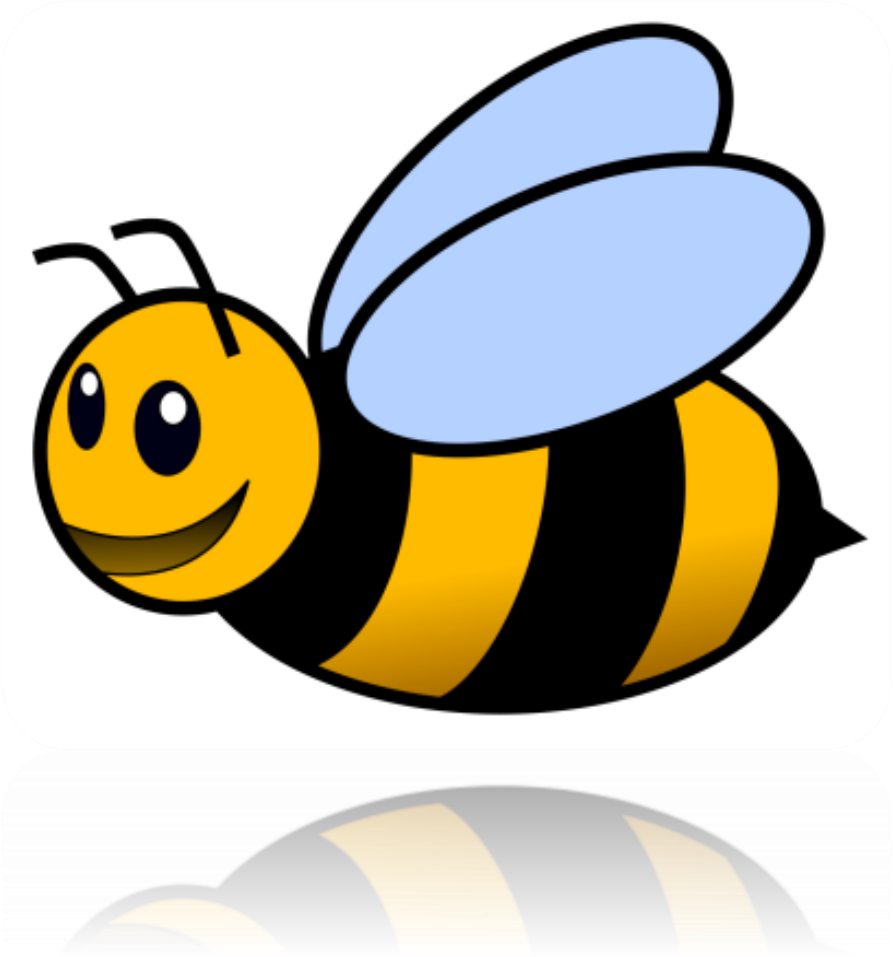
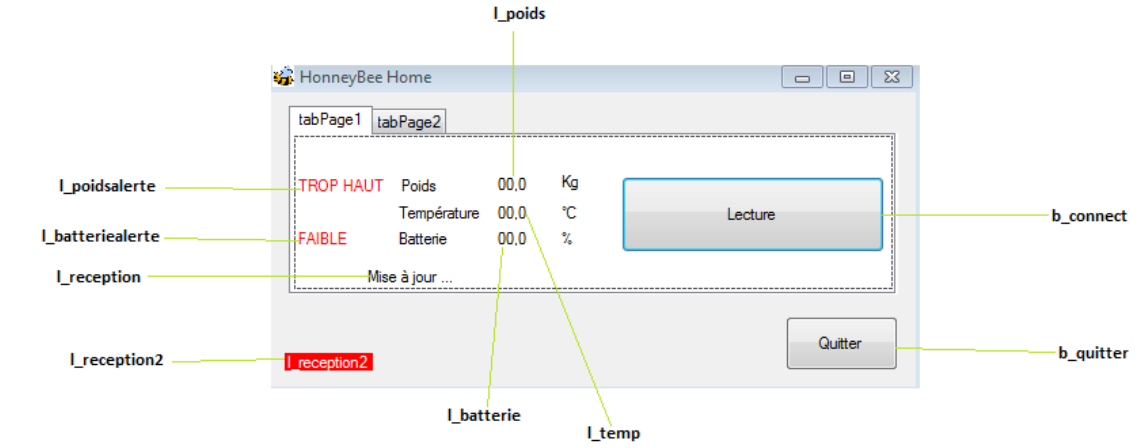
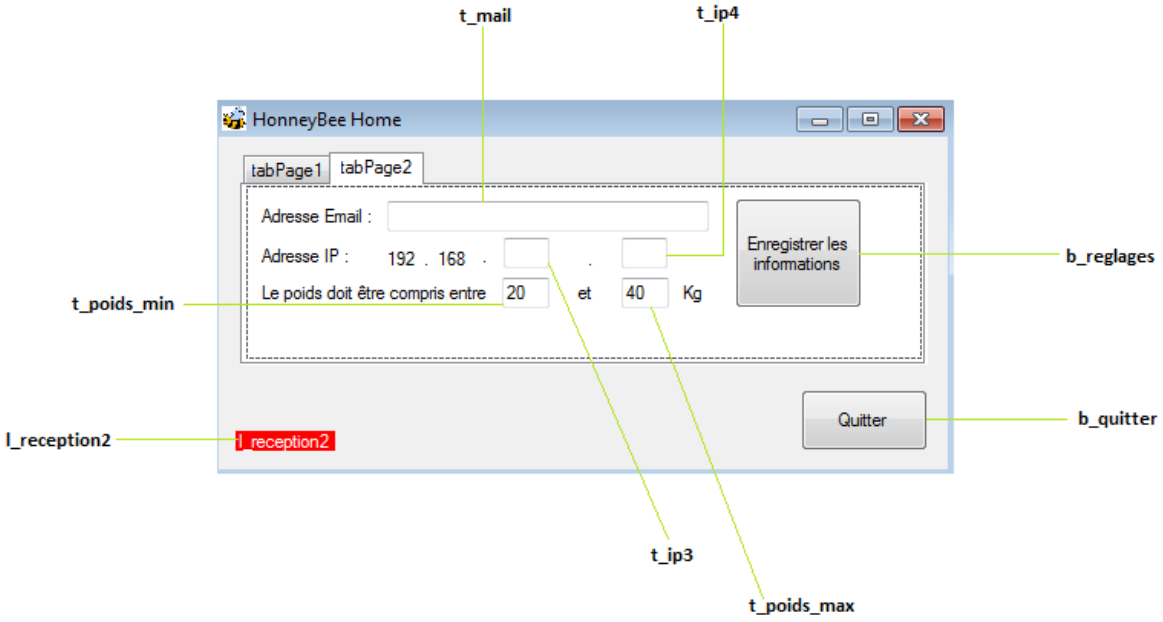


Table des matières

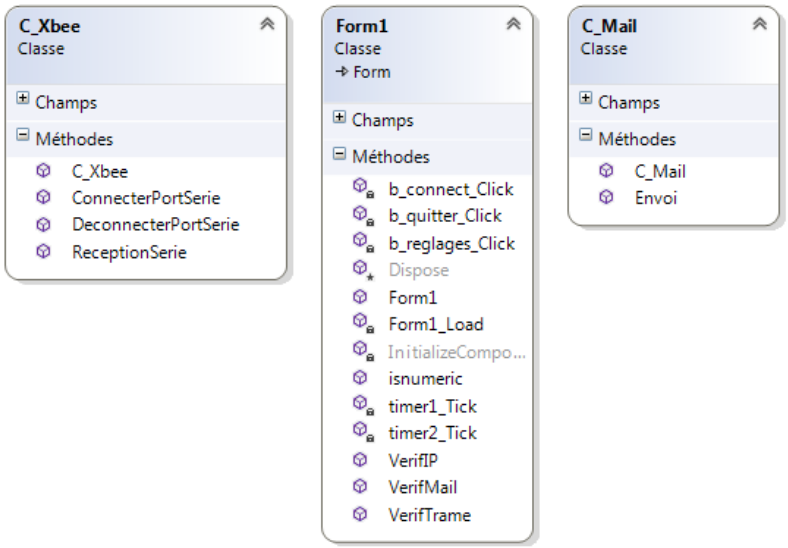
I) Interface homme-machine





II) Les classes

Ce programme dispose de 2 classes : La classe C_Xbee et la classe C_Mail.



I) La classe C_Xbee

La classe C_Xbee gère la connexion du module émetteur et la réception des données.

La méthode ConnecterPortSerie permet d'ouvrir le port série correspondant à celui du module connecté, et renvoie une valeur True ou False en cas d'échec ou de succès de cette opération.

Etant donné que le port utilisé pour le programme (COM20) est virtuel, le port ne pourra pas être ouvert si le module est déconnecté au pc de l'utilisateur :

```
class C_Xbee
{
    SerialPort my_serie = new SerialPort();
    bool EstOuvert = false;

    public C_Xbee()
    {
    }

    public bool ConnecterPortSerie(string portCom, string baudrate)
    {
        my_serie.PortName = portCom;
        my_serie.BaudRate = Convert.ToInt32(baudrate);
        my_serie.DataBits = 8;
        my_serie.StopBits = StopBits.One;
        my_serie.Parity = Parity.None;
        my_serie.Handshake = Handshake.None;
        my_serie.Open();

        if (my_serie.IsOpen == true)
        {
            EstOuvert = true;
        }
        return EstOuvert;
    }
```

La méthode DéconnecterPortSerie permet quant à elle, de fermer le port série et libérer les ressources utilisées précédemment par celui-ci grâce à la méthode Dispose() :

```
public bool DeconnecterPortSerie()
{
    if (my_serie.IsOpen)
    {
        my_serie.Close();
        my_serie.Dispose();
        if (my_serie.IsOpen == false)
        {
            EstOuvert = false;
        }
        else { EstOuvert = true; }
    } return EstOuvert;
}
```

La méthode ReceptionSerie lit les données existantes sur le port série (donc le module) et les retourne, sur un intervalle d'une seconde :

```
public string ReceptionSerie()
{

        Thread.Sleep(1000);//attente pour les gros paquets de données
        string data = my_serie.ReadExisting(); //Lis les données existantes sur le périphérique du port série
        return data; //Retourne ces données
}
```

II) La classe C_Mail

Dans cette classe, nous avons déclaré un constructeur :

```
public C_Mail(string mailexp, string serversmtp, string portsmtp, string usersmtp, string pwdsmtp, bool sslsmtp)
{
    _mailexp = mailexp;
    _serversmtp = serversmtp;
    _portsmtp = Convert.ToInt32(portsmtp);
    _usersmtp = usersmtp;
    _pwdsmtp = pwdsmtp;
    _smtpssl = sslsmtp;
}
#endregion
```

mailexp correspond à l'adresse mail de l'expéditeur.

serversmtp correspond au serveur SMTP utilisé pour envoyer notre mail (dans notre cas, nous utiliserons celui de gmail).

portsmtp correspond au port du serveur SMTP utilisé (port 587).

usersmtp correspond au nom d'utilisateur SMTP (identique à l'adresse mail de l'expediteur).

pwdsmtp correspond au mot de passe de l'adresse mail de l'expéditeur.

Sslsmtp correspond à l'état (activé ou désactivé) du SSL, un protocole de sécurisation des échanges sur Internet. Il sera désactivé dans notre cas.

Il y a également la classe Envoi, qui enverra les mails via le serveur SMTP choisi :

Destinataire correspond à l'adresse mail du destinataire.

Usern correspond à l'adresse mail de l'expéditeur.

Userp correspond au mot de passe de l'adresse mail de l'expéditeur.

Body correspond au contenu texte du mail.

Le Credential permet de définir les informations d'authentification à utiliser.

```
public bool Envoi(string destinataire, string usern, string userp, string body)
{
    try
    {

        MailMessage mail = new MailMessage();
        mail.From = new MailAddress(_mailexp);
        mail.To.Add(new MailAddress(destinataire));
        mail.Subject = "Alerte Honey Bee";
        mail.Body = body;
        SmtpClient SmtpServer = new SmtpClient();
        SmtpServer.Host = _serversmtp;
        SmtpServer.Port = _portsmtp;
        SmtpServer.UseDefaultCredentials = false;
        SmtpServer.DeliveryMethod = SmtpDeliveryMethod.Network;
        SmtpServer.Credentials = new NetworkCredential(_usersmtp, _pwdsmtp);
        SmtpServer.EnableSsl = _smtpssl;
        SmtpServer.Send(mail);
        return true;
    }
    catch { return false; }
}
```

III) Le Main

I) Les méthodes

```
public Form1()
{

        InitializeComponent();
        tabPage1.Text = "Lecture";
        tabPage2.Text = "Réglages";
        l_batteriealerte.Visible = false;
        l_poidsalerte.Visible = false;
        if (ConfigurationManager.AppSettings["adresseip"] != " ")
        {
            string[] decoupeip = adresseipcomplete.Split('.');
            t_ip3.Text = decoupeip[2];
            t_ip4.Text = decoupeip[3];

        }

        if (ConfigurationManager.AppSettings["mail"] != " ")
        {
            t_mail.Text = adressemail; //affiche dans t_mail l'adresse mail enregistrée dans app.config
        }
        if (ConfigurationManager.AppSettings["poidsmin"] != " ")
        {
            t_poids_min.Text = ConfigurationManager.AppSettings["poidsmin"];
            //affiche dans t_poids_min le contenu du réglage "poidsmin" de l'app.config
            poidsmin = Convert.ToInt32(t_poids_min.Text); //Convertion du texte de la boite de texte t_poids_min en entier signé 32 bits
        }
        if (ConfigurationManager.AppSettings["poidsmax"] != " ")
        {
            t_poids_max.Text = ConfigurationManager.AppSettings["poidsmax"];
            //affiche dans t_poids_max le contenu du réglage "poidsmax" de l'app.config
            poidsmax = Convert.ToInt32(t_poids_max.Text); //Convertion du texte de la boite de texte t_poids_min en entier signé 32 bits

        }
```

a) VerifMail

```
public bool VerifMail(string adresse)
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^([\S][^@]+)@([a-zA-Z]+)\.([a-zA-Z]+)$");
    // verifie que l'adresse est sous format: tout caractere alpha numerique sauf @ + @ + chaine de caractere + point + chaine de caractere
    return myRegex.IsMatch(adresse); // retourne true ou false selon la vérification
}
```

b) IsNumeric

```
public bool isnumeric(string chaine)
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^[0-9]+$");
    //Verifie si la chaine est bien un nombre
    return myRegex.IsMatch(chaine); // retourne true ou false selon la vérification
}
```

c) VerifTram

```
public bool VerifTrame(string data)
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^\d{2}\.\d\s\d{2}\.\d\s\d{2}\.\d$");
    //verifie si la trame est sous la forme: deux chiffres + point + chiffre + espace + deux chiffres
    //+ point + chiffre + espace + deux chiffres + point + chiffre
    return myRegex.IsMatch(data); // retourne true ou false selon la vérification
}
```

d) VerifIP

```
public bool VerifIP(string data)
{
    System.Text.RegularExpressions.Regex myRegex = new Regex(@"^\d{3}|\d{2}|\d{1}$");
    //Verifie que l'octet de l'adresse ip est un, doux ou trois chiffres
    if (myRegex.IsMatch(data)) //si le format est respecté
    {
        if (isnumeric(data) == true) //Si la chaine est bien un nombre
        {
            if (Convert.ToInt16(data) > 0 && Convert.ToInt16(data) < 255) // si le nombre < 255
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        else
        {
            return false;
        }
    }
    else
    {
        return false;
    }


}
```

II) Code des boutons et timers

a) B_connect

```
private void b_connect_Click(object sender, EventArgs e)
{
    if (b_connect.Text == "Lecture")
    {
        try
        {
            connect = Xbee.ConnecterPortSerie("COM20", "9600"); //Connexion au port COM20 paramétré au préalable avec une vitesse de
                                                                //9600 bauds
        }
        catch
        {
            MessageBox.Show("Module sans fil récepteur non détecté, vérifiez que la connexion soit effectuée.");
        }
        if (connect == true)
        {
            timer1.Start(); //Demarre le timer qui va executer en continu la lecture des données dans le buffer
            b_connect.Text = "Arrêter";
        }
    }
    else
    {
        connect = Xbee.DeconnecterPortSerie();
        timer1.Stop(); //Arrete le timer et donc arrete la lecture des données du buffer
        if (connect == false)
        {
            b_connect.Text = "Lecture";
        }
    }

}
```

b) B_Reglages

```
private void b_reglages_Click(object sender, EventArgs e)
{
    adresseipcomplete = "192.168." + t_ip3.Text + "." + t_ip4.Text;
    if (VerifMail(t_mail.Text) == true) //Si le format de l'adresse mail est respecté
    {
        adressemail = t_mail.Text;
        //on charge la configuration
        System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
        //On efface la clé voulue dans app.conf
        config.AppSettings.Settings.Remove("mail");
        //On la rajoute dans config.App
        config.AppSettings.Settings.Add("mail", adressemail);
        //On la resauvegarde
        config.Save();
        ConfigurationManager.RefreshSection("appSettings");
        b_reglages.ForeColor = System.Drawing.Color.Black;
    }
    else
    {
        MessageBox.Show("Adresse Email invalide, vérifiez l'orthographe de celle-çi !");
    }
    if (VerifIP(t_ip3.Text) == true && VerifIP(t_ip4.Text) == true) //si le format des octets de l'IP sont bons
    {

        //on charge la configuration
        System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
        //On efface la clé voulue dans app.conf
        config.AppSettings.Settings.Remove("adresseip");
        //On la rajoute dans config.App
        config.AppSettings.Settings.Add("adresseip", adresseipcomplete);
       //On la resauvegarde
       config.Save();
       ConfigurationManager.RefreshSection("appSettings");
       b_reglages.ForeColor = System.Drawing.Color.Black;
   }
   else
   {
       MessageBox.Show("Adresse IP invalide, vérifiez la saisie de celle-çi !");
   }
   try
   {
       //Convertion du texte de la boite de texte t_poids_max en entier signé 32 bits
       poids = Convert.ToDouble(l_poids.Text); //Convertion du texte du label l_poids en entier signé 32 bits
       if (Convert.ToInt32(t_poids_min.Text) > Convert.ToInt32(t_poids_max.Text))
       {
           MessageBox.Show("Seuils de poids invalides, le seuil de poids maximum ne peut pas être inférieur au seuil de poids minimum");
       }
       else if (Convert.ToInt32(t_poids_min.Text) == Convert.ToInt32(t_poids_max.Text))
       {
           MessageBox.Show("Seuils de poids invalides, le seuil de poids maximum ne peut pas être égal au seuil de poids minimum");
       }
       else if (Convert.ToInt32(t_poids_max.Text) > 150 || Convert.ToInt32(t_poids_min.Text) < 0)
       {
           MessageBox.Show("Seuils de poids invalides, les seuils de poids ne peuvent être inférieur à 0kg ou supérieur à 150 kg");
       }
       else if (isnumeric(t_poids_max.Text) == false || isnumeric(t_poids_max.Text) == false)
       {
           MessageBox.Show("Seuils de poids invalides, veuillez entrer des nombres et UNIQUEMENT des nombres s'il vous plait !");
       }
```

```
        else
        {

            System.Configuration.Configuration config = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);
            //On efface la clé voulue dans app.conf
            config.AppSettings.Settings.Remove("poidsmin");
            //On la rajoute dans config.App
            config.AppSettings.Settings.Add("poidsmin", t_poids_min.Text);
            //On efface la clé voulue dans app.conf
            config.AppSettings.Settings.Remove("poidsmax");
            //On la rajoute dans config.App
            config.AppSettings.Settings.Add("poidsmax", t_poids_max.Text);
            //On la resauvegarde
            config.Save();
            ConfigurationManager.RefreshSection("appSettings");
            poidsmin = Convert.ToInt32(ConfigurationManager.AppSettings["poidsmin"]);
            poidsmax = Convert.ToInt32(ConfigurationManager.AppSettings["poidsmax"]);
        }
    }
    catch
    {
        MessageBox.Show("Seuils de poids invalides"); //Message d'erreur en cas de saisie invalide des seuils de poids
    }
```

c) Timer 1

```
private void timer1_Tick(object sender, EventArgs e)
{
    int i;
    string donnees = Xbee.ReceptionSerie();//Récupère les données sous forme "poids temperature batterie"
   if (VerifTrame(donnees) == true)
    {
        l_batteriealerte.Visible = false;
        l_poidsalerte.Visible = false;
        donnees = donnees.Replace(".",","); //remplace les points dans les données par des virgules afin d'éviter les bugs
        l_reception2.Visible = false;
        nbprobleme = 50;
        string[] decoupe = donnees.Split(' '); //découpe la trame pour séparer le poids, la température et la batterie
        l_poids.Text = decoupe[0]; // dispose les différents éléments dans les labels qui leur sont affectés
        l_temp.Text = decoupe[1];
        l_reception.Visible = true; //affiche que les données sont bien mises à jour
        timer2.Start();
        Int32 batteriefaible = 20;
        Int32 batteriemoyenne = 51;
        l_batterie.ForeColor = System.Drawing.Color.Green;
        Double batterie = Convert.ToDouble(decoupe[2]); //Conversion de la batterie en entier pour traiter sa valeur
        Double batteriepourcent = (batterie * 100) / 5; //Conversion de la natterie en pourcentage
        l_batterie.Text = batteriepourcent.ToString();
        Double poids = Convert.ToDouble(l_poids.Text); //Conversion du poids réel en entier pour le comparer aux seuils
        if (batteriepourcent < batteriemoyenne) { l_batterie.ForeColor = System.Drawing.Color.Orange; }
        else if (batteriepourcent == 100) { l_batterie.ForeColor = System.Drawing.Color.Green; }
        string heuredirect = DateTime.Now.ToString("HH:mm:ss");
        string momentjournée = "";
        if (heuredirect == "12:00:00")
        {
            momentjournée = "matin";
            string date = DateTime.Today.ToString("d/M/yyyy");
```

```
    string messageandroid = date + " " + momentjournée + " " + donnees;
    byte[] msg = Encoding.Default.GetBytes(messageandroid); //encode les données en une séquence d'octets
    serveur.Send(msg, msg.Length, adresseipcomplete, 5053);//Envoie les données au smartphone à 21h
}

if (poids < poidsmin)
{
    l_poidsalerte.Text = "TROP BAS"; //Si le poids est en dessous du seuil de poids minimum, alors message d'erreur
    l_poidsalerte.Visible = true;
    l_poids.ForeColor = System.Drawing.Color.Red;
    if (mailpoids == false) //si aucun mail n'a été envoyé depuis le lancement du programme
    {
        try { email.Envoi(ConfigurationManager.AppSettings["mail"],
            "honeybeeparticulier@gmail.com",
            "honeybee201557",
            "Aujourd'hui, le " +DateTime.Now.ToString("dd/M/yyyy hh:mm") + ", le poids est en sous régime (" + poids + " kg)"
            mailpoids = true; } //envoie un mail d'alerte à l'adresse enregistrée
        catch { MessageBox.Show("Echec de l'envoi du mail, vérifiez votre connexion ou l'adresse Email entrée"); }
    }
}
```

```
else if (poids > poidsmax)
{
    l_poidsalerte.Text = "TROP HAUT"; //Si le poids est au dessus du seuil de poids maximum, alors message d'erreur
    l_poidsalerte.Visible = true;
    l_poids.ForeColor = System.Drawing.Color.Red;
    if (mailpoids == false)
    {
        try { email.Envoi(ConfigurationManager.AppSettings["mail"], "honeybeeparticulier@gmail.com", "honeybee201557"
            , "Aujourd'hui, le " + DateTime.Now.ToString("dd/M/yyyy hh:mm") + ", le poids est trop élevé ("
            + poids + " kg)"); mailpoids = true; }
        catch { MessageBox.Show("Echec de l'envoi du mail, vérifiez votre connexion ou l'adresse Email entrée"); }
    }
}
if (batteriefaible > batteriepourcent) //Avertissement si la batterie est à 20% ou inférieure
{
    l_batteriealerte.Visible = true;
    l_batterie.ForeColor = System.Drawing.Color.Red;
    if (mailbatterie == false)
    {
        try { email.Envoi(ConfigurationManager.AppSettings["mail"], "honeybeeparticulier@gmail.com", "honeybee201557"
            , "Batterie faible ! (" + batterie + " %) le " + DateTime.Now.ToString("dd/M/yyyy hh:mm"));
            mailbatterie = true; }
        catch { MessageBox.Show("Echec de l'envoi du mail, vérifiez votre connexion ou l'adresse Email entrée"); }
    }
}
for (i = 0; i < 3; i++)
{
    decoupe[i] = null; //Initialisation du tableau de données
}
```

```
        else
         {
             nbprobleme--;
             if (nbprobleme <= 30)
             {
                 l_reception2.Text = "Mauvaise réception : "+nbprobleme; //affiche un message de mauvaise reception si l'application
                 //n'a pas pu recuperer d'infos 20 fois consecutives
                 l_reception2.Visible = true;
             }
             if (nbprobleme <= 0)
             {
                 timer1.Stop(); // coupe la connexion si la communication entre les modules est mauvaise.
                 timer1.Enabled = false;
                 b_connect.Text = "Lecture";
                 MessageBox.Show("Communication impossible entre les deux modules, vérifiez que les connexions soient effectuées.");
                 try { email.Envoi(ConfigurationManager.AppSettings["mail"],
                     "honeybeeparticulier@gmail.com",
                     "honeybee201557",
                     "Communication impossible entre les deux modules, vérifiez que les connexions soient effectuées.");
                 } // Envoie un mail à l'adresse enregistrée pour informer de la mauvaise communication
                 catch { MessageBox.Show("Echec de l'envoi du mail, vérifiez votre connexion ou l'adresse Email entrée"); }
                 b_connect.Text = "Lecture";

             }
         }
    }
```

d) Timer 2

```
private void timer2_Tick(object sender, EventArgs e)
{
    timer2.Stop();
    l_reception.Visible = false;
}
```