Objectifs :

- Avec C#, découvrir l'espace de noms System.Net et utiliser, entre autre, les classes Ping, Dns et NetworkInterface
- Faire une application de type tableau de bord réseau (*dashboard*) dans lequel on pourra afficher plusieurs informations et lancer plusieurs commandes liées au réseau.
- Avec C#, utiliser les contrôles de données DataGrid et PropertyGrid pour afficher des données tabulaires
- Maitriser les propriétés Dock et Anchor des contrôles WinForm pour positionner les contrôles
- Architecture d'application
  - Créer une bibliothèque dynamique (DLL) en .Net qui pourra être partagée entre une application WinForm et une application Console.
  - Rediriger le flux stdout vers un textbox
  - Découvrir le nouveau modèle de programmation asynchrone `async await` qui remplace les modèles APM et EAP[1]

Directives :

- Le travail peut être fait en équipe de 2 au maximum.
- Nettoyer la solution, zipper (.zip) le tout et remettre selon directive.
- Mettre tous les timeout à 2 secondes pour toute communication réseau.

Étapes :

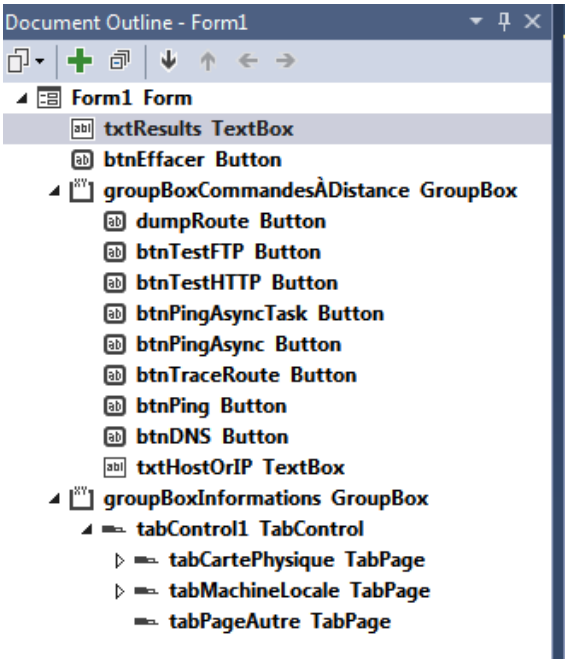
a. Créer un nouveau projet WinForm
b. Regarder l'exemple ExempleListerInterfaces et analyser le code. Analysez plus particulièrement l'utilisation des méthodes Select(), Join() et de la syntaxe LINQ.
c. En vous inspirant de l'exemple ExempleListerInterfaces, créer une méthode qui aura l'en-tête `NetworkInterface TrouverInterfaceRéseauEthernet()`. Utilisez les propriétés de la classe `NetworkInterface` qui vous semble utile pour déterminer quelle est la bonne carte réseau. On cherche celle qui est active et avec laquelle vous êtes connecté sur le réseau du cégep et internet. Ça devrait aussi fonctionner chez vous.
d. (Bonus) Faire la fonction précédente en une seule ligne de code. Vous aurez probablement besoin de IEnumerable.First ou Array.Find.
e. En utilisant des contrôles visuels[2], extraire et afficher l'information pertinente de la carte réseau à partir de celle trouvée précédemment. Affichez au moins les propriétés suivantes :
   - le nom de la carte réseau
   - les statistiques IPv4
   - l'adresse physique MAC (vous avez déjà le code pour formater l'adresse convenablement)
   - la vitesse, le type, le statut opérationnel

[1] https://msdn.microsoft.com/en-us/library/jj152938%28v=vs.110%29.aspx
[2] Label et Textbox sont trop simplistes et lourds à maintenir, utilisez un ListView ou un DataGrid avec 2 colonnes ou un PropertyGrid. Le PropertyGrid n'est pas disponible par défaut, voir comment l'ajouter à la fin du document.

f. Développer la fonction `TesterPing` suivante. Vous devrez utiliser la classe `Ping` et la fonction `Send` qui vous permettra d'obtenir un objet de type `PingReply`. Ajouter des contrôles à l'écran pour pouvoir saisir l'adresse (*textbox*) et appeler votre fonction (*button*). Afficher ensuite le résultat dans un *textbox* multiligne. Voir annexe pour utilisation d'un *textbox* multiligne. Extraire l'information pertinente de l'objet `PingReply` comme l'adresse IP de la réponse[3], temps de réponse et le TTL[4] et l'afficher.

```
/// <summary>
/// Envoie une requête de ping à l'adresse spécifiée
/// </summary>
/// <param name="adresse">Un IP ou un nom de domaine</param>
/// <param name="réponse">Les détails sur la réponse</param>
/// <returns>vrai si le ping est un succès</returns>
bool TesterPing(string adresse, out PingReply réponse)
```

g. Groupez vos deux contrôles (TextBox et le bouton) ensemble dans un GroupBox pour les commandes distantes.
h. Mettez votre contrôle contenant l'information sur la carte physique dans le premier onglet d'un TabControl qui sera lui-même dans un autre GroupBox.
i. Ajoutez un bouton pour vider le TextBox multiligne.
j. Consultez ensuite le "Document outline" de votre *Form.* Pour faire afficher votre *Document outline*, faire CTRL-ALT-T ou passer par le menu VIEW. À noter qu'on peut déplacer des contrôles à même cet fenêtre. Vous devriez avoir 4 éléments sous votre *form*. Voir image exemple →



k. Pour l'instant, vous avez surement positionner les contrôles à l'œil en les redimensionnant manuellement. On veut faire en sorte que les contrôles principaux (les `groupBox`, le `TextBox` et le bouton d'en bas s'ajustent automatiquement à la taille de la fenêtre. Pour se faire, utilisez la propriété DOCK des contrôles. Le TextBox est `dock=fill` et les 2 GroupBox sont `dock=top`. S'ils se chevauchent et deviennent superposés, il faut utiliser l'option de menu contextuel "`Bring to front`" ou "`Send to back`" pour qu'ils se placent.
l. Vous devez configurer les ancrages (propriété `ANCHOR`) du `textBox` et du bouton Ping d'à côté pour que le *`TextBox`* agrandisse avec la fenêtre et que le bouton se déplacent.

[3] pourrait être différente
[4] représente la durée de vie restant du paquet reçu. Elle commence souvent à 64 et diminuée de 1 à chaque fois que le paquet traverse un nœud du réseau (un routeur par exemple)

m. Votre programme devra éventuellement gérer sans problème les cas suivants comme valeurs possibles du TextBox :
   - un IP avec un format invalide
   - un IP inatteignable
   - un nom de domaine invalide
   - un nom de domaine inexistant

   Pour y arriver, dans TesterPing, ajoutez la validation de IP avec `IPAdress.TryParse`. Si ce n'est pas un IP valide faire la validation de la syntaxe de nom de domaine avec `Uri.CheckHostName`. Afficher les erreurs avec `WriteLine`. Vous ne verrez pas les erreurs dans votre TextBox pour l'instant mais plutôt dans VS sous l'onglet Output. On va y remédier…
n. Rediriger le buffer stdout vers votre TextBox en ajoutant la classe fournie TextBoxWriter.cs et en ajoutant la ligne suivante dans l'événement Load de votre fenêtre :
   ```
   Console.SetOut(new TextBoxWriter(txtResults));
   ```
o. Déplacez `TesterPing` dans une classe dans un projet de type "Class library" que vous pouvez appeler UtilitaireRéseau ou un truc du genre. La classe peut être statique et la fonction devra être publique et statique. Votre projet WinForm devra maintenant faire référence à ce nouveau projet DLL pour pouvoir appeler la fonction.
p. Le problème de nom de domaine valide et inexistant lance encore une exception. L'idéal serait de convertir nous même le nom de domaine en IP avec une requête DNS. Pour l'instant, ajoutez un `Try Catch` sur le type `PingException`.
q. Ajouter un *tab* nommé "Machine locale" dans votre TabControl. Ajoutez l'information sur la machine locale en appellant `Dns.GetHostName` pour trouver le nom de la machine et `Dns.GetHostAddresses(…)` pour les adresses IPv4 et IPv6 de la machine.
r. Ajoutez un bouton nommé DNS dans le GroupBox des commandes. Il fera une requête Dns avec Dns.GetHostEntry pour le IP ou le nom de domaine spécifié et fera afficher le nom de domaine résultant et la liste des adresses IP associées.
s. Qu'arrivera-il si la réponse du ping ou de la requête DNS prenait un très long temps à revenir? L'écran deviendra non *responsive*. La bonne pratique est de lancer les appels qui peuvent être potentiellement longs en mode asynchrone. Pour cette étape, nous allons modifier la fonction `TesterPing` (en fait, vous pouvez en créer une nouvelle) qui aura la signature suivante :
   ```
   async public static Task<PingReply> TesterPingAsync(string adresse)
   ```
   - Cette syntaxe utilise la nouvelle technique de programmation aynchrone `async-await` fourni depuis C# 5.0. Le mot clé `async` signifie que cette fonction contiendra probablement des points de suspension (les `await`).
   - Voir ce lien pour un tutoriel vidéo complet sur `async await` si nécessaire (item #14 et 15 de la playlist) : https://www.youtube.com/playlist?list=PLQKPLm6pluP-KGu2u0x1LgiBZ6IkoPfPh
   - Dans votre fonction TesterPing, appeller `SendPingAsync` au lieu de `Send`. Vous verrez que la valeur de retour n'est plus PingResult mais bien Task<PingResult>. Votre code devrait ressembler à :
     ```
     Task<PingReply> tâchePing = pinger.SendPingAsync(adresse);
     réponse = await tâchePing;
     ```
   - Faire `await` sur une tâche permet de suspendre le *thread* et d'attendre que le ping soit terminé. L'avantage de cette syntaxe est qu'en attendant, le contrôle est redonné à l'application qui peut continuer à être *responsive*.

- Faite la même chose sur le *event handler* de votre bouton ping :
  - marquer la fonction btnPing_Click du mot async (on peut le mettre n'importe où)
  - faire `await` sur l'appel de votre fonction TesterPingAsync

t. Implémentation de l'équivalent d'un tracert avec les options `-d et -h 15` (voir aide de tracert)
   - Ajouter un bouton qui lance la trace vers l'adresse ou le IP fourni. Nous utiliserons la surcharge de Send ou SendAsync qui permet de fournir un timeout ainsi que l'option TTL pour le *Time to Live.*
   - Votre code devrait ressembler à ca :

```
Ping envoi = new Ping();
string donnéesPing = "Bonjour il y a quelqu'un?";
PingReply réponse = envoi.Send(hostnameOrIP, 10000,
                    Encoding.ASCII.GetBytes(donnéesPing),
                    new PingOptions(ttl, true));
```

   - Vous pouvez envoyer une sonde seulement par nœud au lieu de 3.
   - Afficher le résultat sous la forme

```
Noeud #1 : Un noeud a été trouvé : 192.168.xxx.xxx
Noeud #2 : Un noeud a été trouvé : 10.248.xxx.xxx
Noeud #3 : Un noeud a été trouvé : 10.170.xxx.xxx
Noeud #4 : Un noeud a été trouvé : 74.116.xxx.xxx
Noeud #5 : La destination 96.127.250.27 est atteinte en 9 ms!
```

u. Ajouter un bouton "Sonde HTTP" et tester s'il y a un service web à l'adresse spécifiée avec la classe HttpClient
   - Lire au besoin : http://www.jayway.com/2012/03/13/httpclient-makes-get-and-post-very-simple/
   - Le code suivant permettra de lancer la requête HTTP de façon asynchrone. Vous pouvez utiliser `await` seulement dans une fonction `async` alors vous devez marquer la fonction courant (l'événement on click du button) du mot clé `async`.

```
HttpClient httpClient = new HttpClient();
httpClient.BaseAddress = new Uri("http://" + txtHostOrIP.Text);
HttpResponseMessage réponse = await httpClient.GetAsync("");
réponse.EnsureSuccessStatusCode();
```

   - Le URI représente l'adresse du serveur web.
   - Attention, EnsureSuccessStatusCode lancera une exception si aucun serveur ne répond. Gérer ce cas convenablement.
   - Afficher le code de réponse et la version du protocole HTTP utilisée.

On demande la page racine du site. Souvent, ce sera index.html ou .php ou .aspx par défaut si rien n'est spécifié

v. Faire la même chose pour voir s'il y a un service FTP à cette adresse.
   - Utiliser la classe FtpWebRequest avec la méthode statique de fabrique `FtpWebRequest ftpWr = (FtpWebRequest)FtpWebRequest.Create("ftp://" + txtHostOrIP.Text);`
   - Spécifier quelle méthode FTP sera exécutée si la connexion réussie. On veut lister le répertoire racine : `ftpWr.Method = WebRequestMethods.Ftp.ListDirectory;`

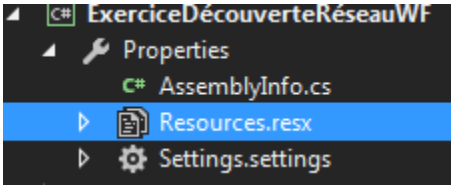
- Exécuter la requête en tentant d'obtenir la réponse `FtpWebResponse réponse = (FtpWebResponse)ftpWr.GetResponse();`
- Si l'adresse est ne représente ni un IP ni un domaine, vous aurez une exception de type UriFormatException. Gérez ce cas et testez-le en mettant des caractères invalides dans l'adresse.
- Il y a plusieurs cas ou vous obtiendrez une exception de type WebException. Ajouter un catch pour ce type.
- Dans ce catch, faites un switch case sur la propriété Status de l'exception et insérer le code suivant

```
case WebExceptionStatus.ProtocolError:
  FtpWebResponse réponse = (FtpWebResponse)ex.Response;
  WriteLine("Présence d'un serveur FTP positive mais erreur de protocole");
  WriteLine(réponse.BannerMessage);
  WriteLine("Code réponse FTP : " + réponse.StatusCode);
  break;
case WebExceptionStatus.NameResolutionFailure:
  WriteLine("Impossible de résoudre le IP");
  break;
default:
  WriteLine("Impossible de contacter un serveur FTP pour une raison non diagnostiquée");
  break;
```

- Si vous tentez sur le serveur FTP du cégep ftp1.cstjean.qc.ca, vous devriez avoir une erreur de protocole avec un code d'erreur FTP 530 qui signifie que vous n'êtes pas authentifié.
- À titre informationnel, pour s'authentifier, il faudrait fournir les *credentials* avant le lancer la requête. `ftpWr.Credentials = new NetworkCredential("cstjean.qc.ca\\<votre_da>", "<votre_mdp>");`

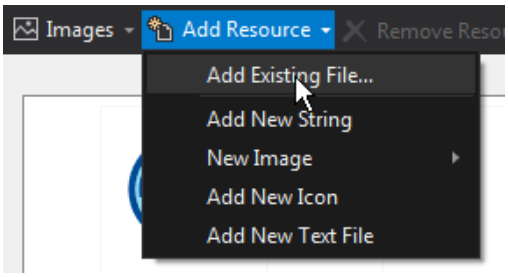
w. Modifier votre test de FTP pour qu'il soit asynchrone en utilisant `GetResponseAsync` au lieu de `GetResponse`.

x. Implémenter un indicateur de progrès circulaire (un *throbber*) pour votre bouton de test HTTP. Si vous cherchez comment faire sur internet, vous verrez plein d'exemples complexes qui requièrent parfois des DLL, des UserControl et des bouts de code de plusieurs centaines de lignes de code[5]. Pourtant, une technologie vieille comme l'internet permet déjà de faire ce genre de chose : le GIF animé.
   - Trouvez-vous un *throbber* mignon ou faites le vôtre à l'adresse http://www.chimply.com/Generator#classic-spinner,animatedCircle
   - Attention, assurez-vous que votre gif animé supporte la transparence sinon ca va être laid. Assurez-vous qu'il soit 16px par 16px comme résolution native.
   - Il faut maintenant ajouter votre image comme ressource du projet.
   - Ouvrez le gestionnaire de ressources globales de votre projet. C'est le fichier *.resx* sous le nœud *Properties*.



[5] Traduction d'un paragraphe sur http://trompelecode.com/blog/2010/12/animated-progress-indicator-in-csharp-windows-forms/

- Dérouler le bouton *Add resource* et choisir *Add Existing File*



- Vous pouvez maintenant mettre le code suivant au début de l'événement du bouton :

```
btnTestFTP.Text = "EN COURS";
btnTestFTP.Image = Properties.Resources.animatedEllipse16px;
```

- Et à la fin de l'événement (dans un finally?)

```
btnTestFTP.Text = "Test FTP";
btnTestFTP.Image = null;
```

> Tout n'est pas encore parfait. Il faudrait empêcher l'usager de cliquer le même bouton pendant que l'opération est en cours. En fait, dans un monde idéal, il faudrait offrir la possibilité d'annuler toute commande lancée de façon asynchrone au lieu d'attendre le timeout.

y. Mettre un bouton "Info MAC" sous le tab de la carte physique. Ce bouton obtient le nom du fabricant de la carte réseau auprès du service Web http://api.macvendors.com/.

- Pour y parvenir, faites tout simplement une requête HTTP vers cette adresse en ajoutant l'adresse MAC formattée :

```
httpClient.BaseAddress = new Uri("http://api.macvendors.com/" + macFormattée );
```

- Vous devez maintenant obtenir la réponse de la même façon vue précédemment
- Et extraire le contenu de la réponse, ce sera le fabricant

```
string fabricant = await réponse.Content.ReadAsStringAsync();
```

Annexe

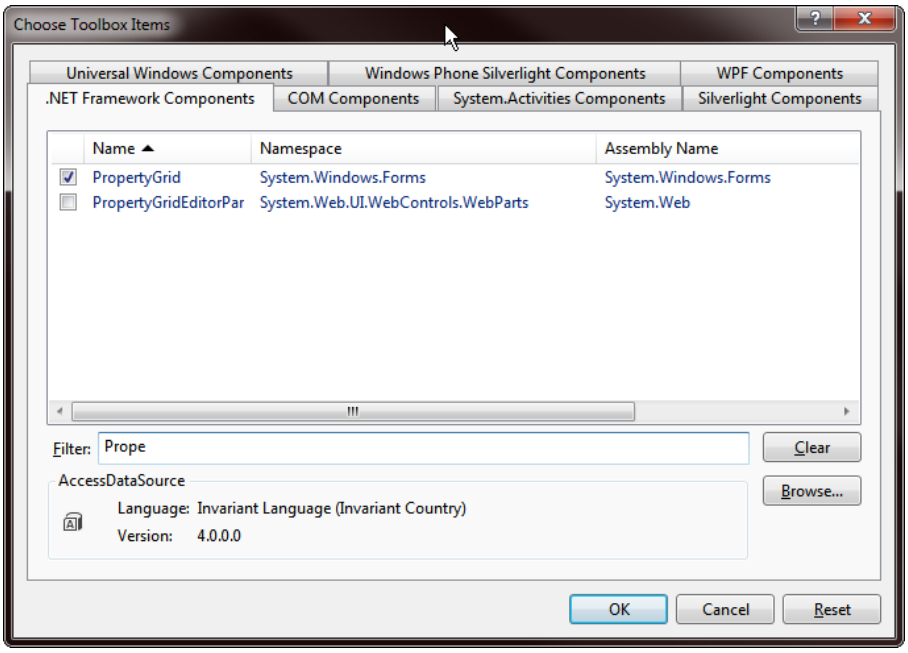
PropertyGrid

Pour ajouter le contrôle PropertyGrid, aller dans "Tools/Customize toolbox items" et choisir l'onglet "NET Framework components". Saisir "PropertyGrid" dans le champ de filtre. Cocher l'item pour Winform. Cliquer ok et il devrait être disponible dans la boite d'outils.

https://msdn.microsoft.com/en-us/library/aa302326.aspx#usingpropgrid_topic1

Pour remplir le PropertyGrid, ne cherchez pas l'option pour insérer des valeurs. Il faut utiliser le DataBinding, une option qui permet de spécifier un objet et le propertyGrid se remplira lui-même. Il suffit de "setter" la propriété SelectedObjet comme suit :

```
propertyGrid1.SelectedObject = NetUtils.TrouverInterfaceRéseauEthernet();
```



Textbox multiligne

On peut configurer un simple *textbox* en zone d'affichage pour un journal de messages. Pour cela, il faut utiliser `TextBox.AppendText` et `Environment.NewLine` pour ajouter des lignes dans le `TextBox`. Il faut avoir défini les propriétés suivantes au préalable :

- ScrollBars à Vertical
- ReadOnly à true
- Multiline à true
-

Autre façon de faire : http://www.csharp-examples.net/autoscroll/