GB とか KB とかを全部 MB にする

何を言ってるか分からないと思うが、とにかく 10GB とか 5000KB とかそういう文字列を全部 MB 単位に合わせる。
単位に合わせて MB とか GB とか付けるのの逆をやる。

動作例

```
4GB → 4096
256 → 256
1024KB → 1
10G → 10240
```

コード

```
private static int ExParse(string moji)
{
    string m = moji.Trim().ToUpper();
    int result = 0;
    if (m.Contains("KB") || m.Contains("K"))
    {
        m = m.Replace("KB", "").Replace("K","").Trim();
        result = (int)(float.Parse(m) / 1024);
    }
    else if (m.Contains("MB") || m.Contains("M"))
    {
        m = m.Replace("MB", "").Replace("M","").Trim();
        result = (int)(float.Parse(m));
    }
    else if (m.Contains("GB") || m.Contains("G"))
    {
        m = m.Replace("GB", "").Replace("G","").Trim();
        result = (int)(float.Parse(m) * 1024);
    }
    else if (m.Contains("TB") || m.Contains("T"))
    {
        m = m.Replace("TB", "").Replace("T","").Trim();
        result = (int)(float.Parse(m) * 1024 * 1024);
    }
    else
    {
        result = (int)(float.Parse(m));
    }

    return result;
}
```

PowerShell

複数のホストに Ping 打ち続ける

１つのホストだけなら

```
ping -t host1
```

みたいな感じで延々と ping を打てるが、ping を打ちたいマシンが複数ある場合はちょっとむずかしい。

なので、PowerShell でやってみた。

```
 $l=@("host1","host2","host3");while(1){$r=$(get-date).DateTime.ToString();$r+="`n";foreach($a in $l){$r+=ping $a -n 1|select-string -Pattern "Reply|Request";$r+="`t$a`n"};$r+="----";$r;Start-sleep -s 1;$n=ipconfig /flushdns}
```

host1, host2, host3 の所に、ホスト名か IP を入れていく。必要に応じて増やす。
実行すると

```
----
2012 年 8 月 3 日 14:06:06
Reply from 172.16.1.1: bytes=32 time<1ms TTL=64
Reply from 172.16.1.2: bytes=32 time<1ms TTL=64
Reply from 172.16.1.3: bytes=32 time<1ms TTL=64
----
```

こんな感じで、ずっと Ping が実行され続ける。

秒の数字を、フォーーーーーマット

```
int time = 123456;
string.Format("{0:00}:{1:00}:{2:00}", (time / 60 / 60), (time / 60) % 60, time % 60)
```

秒を表す数字を、時間：分：秒　にフォーマットする。
時間は、24 時間を越えてもおｋ。

探せば、もっとマシなやり方があるかも。

Visual Studio で作った、セットアップファイルの、対応バージョンを広げる。

Visual Studio でセットアップファイルを作ると、
セットアップ時に 作成した Visual Studio のバージョンに応じた .NET Framework のランタイムが必要になる。

例えば、Visual Studio 2003 で作成したセットアップを、.NET Framework 1.1 が入っていないマシンで実行すると、以下のようなメッセージが表示される。

```
このセットアップは、.NET Framework バージョン 1.1.4322 を必要とします。
.NET Framework をインストールして、このセットアップをやり直してください。
.NET Framework は Web から取得できます。今すぐ取得しますか？
```

もし、セットアップを実行したマシンに .NET Framework 2.0 が入っていて、とりあえずつべこべ言わずにセットアップさせたい場合は、セットアップにサポートするバージョン番号を追記してやる事で、対応できる。

具体的には…
セットアッププロジェクトを右クリックして「表示」→「起動条件」
起動条件が表示されるので、その中から、
「対象コンピュータ上の必要条件」→「起動条件」→「.NET Framework」をクリック。
プロパティを開き、「SupportedRuntimes」に

```
1.1.4322;2.0.50727;3.0.4506
```

という値を設定する。

これによって、.NET Framework が、どれか入っていればインストール出来るようになる。

この値は、1.1、2.0、3.0 をサポートしてますよー。という意思表示なのだが・・・
本当に、全バージョンで動くかどうかは、ちゃんと確認しないとアレです。

ちなみに、1.1 で作ったものは、3.0 でも、基本的には動くが、3.0 で作ったものは 1.1 では動かない。

Win32 のエラーコードを、メッセージに変換する。

まんまです。
FormatMessageA とか、その辺を使えばいいのかな？と思ったのですが、
すげー面倒そうなので、他を調べたら出てきました。

動いたのでメモ。

```
/// <summary>
/// 指定されたエラーコードから、エラーを説明するメッセージと 16 進数表現を返します。
/// </summary>
/// <param name="code"></param>
/// <returns></returns>
public static string FormatMessage(int code)
{
    byte[] b = BitConverter.GetBytes(code);
    Array.Reverse(b);// リトルエンディアンなので
    string bs = "0x" + BitConverter.ToString(b, 0).Replace("-", "");

    return string.Format("{0}({1})", new System.ComponentModel.Win32Exception(code).Message, bs);
}
```

USB 機器を、C# で取り外しを行う

http://www.codeproject.com/csharp/usbeject.asp

ここのコードを参考にしたら、USB デバイスの列挙とか取り外しが、簡単に出来そう。
ユーザー登録しないと、コードダウンロードできないけど。

--- 追記
USB メモリーなどは、取り外しできるけど、
その他の USB デバイスは、このコードのままじゃ取り外しできないっぽい。

実行中プログラムのディレクトリ取得

実行中のプログラムのディレクトリを取得します。
作業ディレクトリとは別物です。

```
public static string _AssemblyDir = Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location);
```

Process クラス周りの標準入出力について

ProcessStartInfo の、RedirectStandardOutput、RedirectStandardInput、とかを設定してやって、コマンドラインプロセスとやりとりをすることが出来る。
で、その際のメモ。

```
Process p = Process.Start(info);
StreamReader reader = p.StandardOutput;
StreamWriter writer = p.StandardInput;
```

まぁ、こんな感じで、リーダー・ライターを用意して、

```
StringBuilder sb = new StringBuilder();
int buffsize = 1024;
char[] buff = new char[buffsize];

int len = 0;

Debug.WriteLine("read");
while ((len = reader.Read(buff, 0, buffsize)) > 0)
{
    sb.Append(buff, 0, len);
    if (sb.ToString().EndsWith(">"))
    {
        writer.WriteLine(...); // コマンド？とかの処理
    }
}
```

てな具合で書き込んでやればいいかな？と。
ちなみに、len が 0 になるのはプロセスが終了したときっぽい。

もし、プロセスが終了していなくて、かつ標準出力に何も出力されていなかった場合、Read であっても、何か出力されるまでブロックされるようだ。

ReadBlock は、バッファがいっぱいになるまでブロックしてそう。試してないけど。

リソースからのデータの読み取り

リソースの追加

プロジェクトを右クリックし、追加、既存の項目
そして、ファイルの種類をすべてのファイル、にして、アセンブリファイルに埋め込みたいファイルを選択。
プロジェクトに追加されただけだと何もしないので、追加されたファイルを選択し、ビルドアクションプロパティを、埋め込まれたりソースにする。

コード

```
using System;
using System.IO;
using System.Reflection;


public static Stream GetAssemblyStream(string name)
{
    Stream stream;

    // アセンブリから読み込む努力。
    Assembly asm = Assembly.GetExecutingAssembly();
    //Assembly asm = Assembly.GetAssembly(typeof(Program)); // 自分のクラスを指定
```

```
        // 何もせずにアセンブリから取れたなら返す。
        stream = asm.GetManifestResourceStream(name);
        if (stream != null)
        {
            return stream;
        }

        // パスとかいじってアセンブリから取ろうとする。
        string fullname = asm.FullName;
        string[] asmname = fullname.Split(",".ToCharArray());

        if (asmname.Length >= 2)
        {
            name = name.Replace("/", ".").Replace("\\", ".");
            name = asmname[0] + "." + name;
            stream = asm.GetManifestResourceStream(name);

            return stream;
        }

        // どれもダメだったら null
        return null;
    }
```

レジストリの書き込み

```
using Microsoft.Win32;


            RegistryKey                                    regkey                                    =
Registry.CurrentUser.CreateSubKey(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Run");
 regkey.SetValue("notepad");
 regkey.Close();
```

文字列の類似度

Levenshtein Distance Algorithm: C# Implementation

ttp://www.merriampark.com/ldcsharp.htm

２つの文字列の距離（違い具合）を数値化する。

```
 （例）
文字１：マイクロソフト
文字２：マイクロシフト
距離：１
```

PadRight は２バイト文字に対応していない？

まぁ、表題の通りなんだけど。

で、その対応したメソッド

```
private static string PadRight(string str, int count)
{
    int c1 = Encoding.Default.GetByteCount(str);
    int c2 = str.Length;
    count += c2 - c1;
    return str.PadRight(count);
}
```

イベントはデリゲートの重複登録を許してるので

自力で、イベントからデリゲートを取り出し、重複してるデリゲートを省いた上でイベントに登録しなおす。

という微妙なコード（抜粋）

重複登録が出来ないようになってりゃいいと思うんだけど、どうなんだろう。

```
public event TestDelegate TestEvent;
public void call(string message)
{
    try
    {
        Delegate[] dgate = TestEvent.GetInvocationList();
        ArrayList list = new ArrayList();
        for(int i=0; i<dgate.Length; i++)
        {
            if(list.Contains(dgate[i]))
            {
                Console.WriteLine("Constains!");
            }
            else
            {
                list.Add(dgate[i]);
            }
        }
        TestEvent = null;
        foreach(object obj in list)
        {
            TestDelegate td = (TestDelegate)obj;
            TestEvent += td;
        }
    }
    catch{}
    if(TestEvent != null)
    {
        TestEvent(message);
    }
}
```

ちなみに、イベントを全部削除するには、内部から null を代入するしかない（たぶん）
外部から出来るのは Event -= new MyDelegate(myMethod); とか、そういう感じでの削除だけ。

ついでに言うと、デリゲートの削除は、一度作ったデリゲートでも、新しく作ったデリゲートでも同じ。

つまり、

```
hoge.HogeEvent += new HogeDelegate(myMethod);
```

このようにして登録したデリゲートを消す場合、

```
hoge.HogeEvent -= new HogeDelegate(myMethod);
```

このコードでも消せる。

```
HogeDelegate hd = new HogeDelegate(myMethod);
hoge.HogeEvent += hd;
hoge.HogeEvent -= hd;
```

こんな具合に消しても同じ。

ただ、最初に書いたように、重複登録できるので、

```
HogeDelegate hd = new HogeDelegate(myMethod);
hoge.HogeEvent += hd;
hoge.HogeEvent += hd;
hoge.HogeEvent -= hd;
```

とすると、hoge.HogeEvent には、一つだけデリゲートが残る。

上でも書いたが、デリゲートをすべて消すには、

```
class HogeClass{
    public event HogeDelegate HogeEvent;
    public void eventClear(){
        HogeEvent = null;
    }
}
```

こんな感じにしといて、null を代入しないと無理っぽい？
外から HogeEvent にイコールで値を代入出来ないので。

よーく注意しなきゃいけないのは、イベントに登録したデリゲートは削除しないと、デリゲートの中身のメソッドを持ってるクラスがガーベッジコレクタに回収されないって事。
当たり前だけど参照がある以上、回収されない。
うっかり削除し忘れるとエライ事に・・・

まぁ普通は、イベントの登録と削除を繰り返すようなプログラムは、あんまり無いかもしれないけど。

値から Enum に変換

```
public enum TestEnum
{
    Unkown,
    Hoge,
    Hoge2,
}
```

こんな列挙型があったとして、

■数値→列挙型

```
TestEnum e = (TestEnum)1;
```

ただし、列挙型の中に無い数値を指定すると、指定した数字がそのまま出てくる。
default の処理は必須。

■文字列→列挙型

```
TestEnum e = (TestEnum)Enum.Parse(typeof(TestEnum), "Hoge");
```

文字列を列挙型に変換出来るが、このときの文字列は大文字小文字を間違えてはいけない。間違えると例外がスローされる。
大文字小文字を間違えてはいけない理由としては、列挙型に定義する文字列自体が、大文字と小文字を区別しているから。
だと思う。

■そのた

列挙型の中に無い値を指定すると（例で言うと、-1 とか 3 とか）数字がそのまま出てくる。
なんじゃこりゃ、って感じだが、とりあえず switch 文で default を書いとけって事か？

```
public static TestEnum ToTestEnum(int a)
{
    TestEnum e = (TestEnum)a;
    try
    {
        if(a == int.Parse(e.ToString()))
        {
            return TestEnum.Unkown;
        }
    }
    catch{}
    return e;
}
```

こんな風にすれば、不明な Enum 番号を Unkown に出来るが…うーん。
default で処理した方がマシか。

.NET のデコンパイラ Reflector

解説サイト
http://www.atmarkit.co.jp/fdotnet/tools/dotfuscator/dotfuscator_01.html

Reflector の配布サイト
http://www.aisto.com/roeder/dotnet/

ダウンロードしようとすると、名前とメールアドレスの入力を要求されるが、適当な情報を入力してもダウンロードできる。

Reflector のアドオン
http://www.denisbauer.com/NETTools/FileDisassembler.aspx

これを、Reflector に登録すると、アセンブリファイルを一発でプロジェクトファイルに変換できる。

特定の DateTime と今日の間の日数を表示 (小数点１桁付き)

```
DateTime dt  = DateTime.Parse("2006/1/1");
DateTime now = DateTime.Now;

double result = Math.Round( ((TimeSpan)(dt-now)).TotalDays, 1);

Console.WriteLine("{0} - {1}", dt, now);
Console.WriteLine("{0}", result);


> 2006/01/01 0:00:00 - 2005/12/02 9:55:42
> 29.6
```

CPU メーター作るぜ

```
using System.Diagnostics;


PerformanceCounter processCounter;
processCounter = new PerformanceCounter("Processor", "% Processor Time", "_Total", true);


int val = (int)processCounter.NextValue();
```

この辺のクラスとかメソッドとかを使えば出来そうな気がする。

宇宙仮面さんのところとか参考になります。
ttp://ukamen.hp.infoseek.co.jp/Programming2/PerfMeter/index.htm

実行中のマシンの LDAP パスの作り方

実行中のマシンから・・・とは書いたが、正確には、ドメインに所属したユーザーがログインしたセッション上から実行しないといけない。
そうじゃないとドメイン情報などの環境変数が取れないので。

ローカルマシンにログインしても、以下のコードじゃ LDAP は作れない。

他に方法があると思うけど、知らない。
（AD の中に、LDAP のリストがあった気もする…？）

```
string userdomain = string.Empty;
string userdns = Environment.GetEnvironmentVariable("userdnsdomain");

if(userdns != null)
{
  string[] tmp = userdns.Split(".".ToCharArray());
  int len = tmp.Length;
  for(int i=0; i<len; i++)
  {
    if(i>0)
    {
      userdomain += ",";
    }
    userdomain += "DC="+tmp[i].ToUpper();
  }
}

string logonserver = Environment.GetEnvironmentVariable("logonserver");
if(logonserver != null)
{
  logonserver = logonserver.Substring(2);
}

textBox3.Text = userdns;
textBox4.Text = "LDAP://"+logonserver+"/"+userdomain;
```

XML ドキュメントファイルの出力

ソリューションエクスプローラからプロジェクトを右クリックし、「プロパティページ」を表示する。
その中の「構成プロパティ」の「ビルド」を選択し、「XML ドキュメントファイル」のところにファイル名を入れて OK を押す。

ビルドすると XML ドキュメントファイルが出力される。
このとき、メソッドなどにコメントを書いていないと
「公開されている型またはメンバ（中略）の XML コメントがありませんん。」

と警告が出る。

この警告を表示しないようにするには、上で表示したプロパティページで、
「特定の警告を表示しない」のところに「1591」と入力する。
すると警告が表示されなくなる。

システム色

```
using System.Drawing


// テキスト色
Color text = SystemColors.WindowText;
// 背景色
Color bg   = SystemColors.Window;


// テキスト色（選択）
Color text = SystemColors.HighlightText;
// 背景色（選択）
Color bg   = SystemColors.Highlight;
```

同期

同期メソッド

```
using System.Runtime.CompilerServices;


[MethodImpl(MethodImplOptions.Synchronized)]
public void hoge(){
    .....
}
```

数字→文字列への変換速度

ただ単に、数字を文字列に変換して、どれが早いか調べてみた。
連結速度では無い点に注意。

早い順（たぶん）

```
(1234).ToString();
Convert.ToString(1234);
1234+string.Empty;
1234+"";
```

問題外

```
StringBuilder sb = new StringBuilder();
sb.Append(1234);
sb.ToString();
```

文字列←→バイト列の相互変換

```
using System.Text;


// インスタンスの生成
```

```
Encoding enc = Encoding.GetEncoding("Shift_JIS");

// バイト列に変換
byte[] bytes = enc.GetBytes("てすと");

// バイト列から文字列に変換
string str = enc.GetString(bytes);
```

使えるエンコーディング名

ttp://www.atmarkit.co.jp/fdotnet/dotnettips/013enumenc/enumenc.html

URL エンコード・デコード

```
using System.Web;
using System.Text;


string target = "ほげほげ";

// UTF-8 で URL エンコード
string result = HttpUtility.UrlEncode(target, Encoding.UTF8);
// UTF-8 で URL デコード
target = HttpUtility.UrlDecode(result, Encoding.UTF8);
```

処理時間の計測

```
sTime = DateTime.Now;
for(i=0;i<Loops;i++) sDest += sSource;
eTime = DateTime.Now;
Console.WriteLine("連結に " + (eTime - sTime).TotalSeconds + " 秒かかりました。");
```

ここからコピってきた。

ttp://support.microsoft.com/default.aspx?scid=kb;ja;306822

処理時間の計測２

System.Environment.TickCount

ここに OS が起動してからの時間がミリ秒で入っているらしい。

精度は 15ms 程度？

```
int sTime = System.Environment.TickCount;

( 何か時間のかかる処理 )

Console.WriteLine("time : "+(System.Environment.TickCount-sTime)+" ms");
```

処理時間の計測３

```
using System.Collections;


private Hashtable timetable = new Hashtable();
private void start(string name)
{
    timetable[name] = System.Environment.TickCount;
}
private void end(string name)
{
    Object o = timetable[name];
    if(o == null)
```

```
        {
            return;
        }
        int t = (int)o;
        Console.WriteLine(name+" : "+(System.Environment.TickCount - t));

        timetable.Remove(name);
    }
```

なんか、こんなカンジのコードを使って、

```
start("test1");
//... 時間のかかる処理
end("test1");
```

とすると、

```
test1 : 16
```

などと、start が呼ばれてから end が呼ばれるまでの時間がミリ秒で表示される。

メリットとしては、計測がネストしても問題ないこと。

```
start("A");
// 時間のかかる処理
start("B");
// 時間のかかる処理
end("B");
// 時間のかかる処理
end("A");
```

当たり前だが、こんな事しても問題ない。

・・・そういや、こんな事しなくても、何かちゃんとした方法があったような無かったような・・・

サービスの開始

servicename という名前のサービスを開始するサンプル。

```
using System.Diagnostics;


ProcessStartInfo info = new ProcessStartInfo("net", "start servicename");
info.CreateNoWindow = true;
info.UseShellExecute = false;

Process p = Process.Start(info);
try
{
  p.WaitForExit();
}
catch{}
```

一時的なメモ

```
static void Main(string[] args)
```

```
{
    Bitmap bmp = new Bitmap("test.png");
    bmp = ResizeImage(bmp, 96, 96);

    ImageCodecInfo jpgEncoder = null; // JPEG 用エンコーダ
    foreach (ImageCodecInfo ici in ImageCodecInfo.GetImageEncoders())
    {
        Console.WriteLine(ici.CodecName);
        if (ici.FormatID == ImageFormat.Jpeg.Guid)
        {
            jpgEncoder = ici;
        }
    }

    EncoderParameter encParam1 = new EncoderParameter(Encoder.Quality, 45L);
            EncoderParameter    encParam2    =    new    EncoderParameter(Encoder.Compression,
(long)EncoderValue.RenderProgressive);

    EncoderParameters encParams = new EncoderParameters(2);
    encParams.Param[0] = encParam1;
    encParams.Param[1] = encParam2;

    bmp.Save("output.jpg", jpgEncoder, encParams);
}

public static Bitmap ResizeImage(Bitmap image, double dw, double dh)
{
    double hi;
    double imagew = image.Width;
    double imageh = image.Height;

    if ((dh / dw) <= (imageh / imagew))
    {
        hi = dh / imageh;
    }
    else
    {
        hi = dw / imagew;
    }
    int w = (int)(imagew * hi);
    int h = (int)(imageh * hi);

    Bitmap result = new Bitmap((int)w, (int)h);
    Graphics g = Graphics.FromImage(result);
    g.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
    g.DrawImage(image, 0, 0, w, h);

    return result;
}
```

アレなメモ

```
//System.Net.ServicePointManager.CertificatePolicy = new CertPolicy();
      System.Net.ServicePointManager.ServerCertificateValidationCallback          =          new
System.Net.Security.RemoteCertificateValidationCallback(CertificateCallback);

  bool    CertificateCallback(object    sender,    System.Security.Cryptography.X509 Certificates.X509
Certificate    certificate,    System.Security.Cryptography.X509 Certificates.X509 Chain    chain,
System.Net.Security.SslPolicyErrors sslPolicyErrors)
{
  return true;
}
```

ストリームの中身を全部読み込んでバイト列にして返す

まんまです。

何バイトのストリームであろうが、読み込んで byte[] に変換して返します。

```
static byte[] ReadAllByte(System.IO.Stream stream)
{
    System.IO.MemoryStream result = new System.IO.MemoryStream();

    int i=0;
    int buffsize = 1024 * 1024;
    byte[] buff = new byte[buffsize];
```

```
    while ((i = stream.Read(buff, 0, buffsize)) > 0)
    {
        result.Write(buff, 0, i);
    }

    result.Seek(0, System.IO.SeekOrigin.Begin);
    return result.ToArray();
}
```