	ARTICOLO SULLE DLL	*date:* ***22/02/2012***

A CURA DELL'ING. BUTTOLO MARCO

SOMMARIO:

Contenuti

INTRODUZIONE:

Scopo di questo documento è quello di descrivere in linea di massima il funzionamento di una DLL sviluppando successivamente una DLL di esempio.

DESCRIZIONE DELLA TECNOLOGIA:

DLL è l'acronimo di **Dynamic Link Library**. In poche parole, una DLL è una libreria software che viene automaticamente caricata in fase di esecuzione. Sono facilmente riconoscibili in quanto DLL è anche l'estensione dei file contenenti tali librerie. Nel mondo Linux tali librerie esistono e vengono chiamate **shared object** (file con estensione .SO). Cerchiamo di capire il funzionamento di una DLL tramite un banalissimo esempio. Supponiamo di voler creare un programma che visualizza a video un messaggio. Ci sono due strade: una strada che porta a sviluppare un programma che da zero visualizzi il messaggio, oppure un'altra strada che consiste nel sviluppare un programma che si appoggia ad una DLL la quale contiene una funzione che effettua la visualizzazione del messaggio. La seconda strada, in casi complessi, può risultare comoda in quanto è sufficiente importare la DLL che fa già lei la computazione ed il gioco è fatto. L'uso delle DLL permette di non dover ogni volta riscrivere le funzioni. In questo modo lo sviluppo dell'applicativo diventa una sorta di assemblaggio di librerie e componenti già pronte all'uso. Il seguente esempio mostra il codice sorgente di una libreria DLL realizzata in Visual Basic .NET:

```
Option Explicit On
Public Class Class1
    Public Sub visualizza()
        MsgBox("Test su DLL!!!")
    End Sub
End Class
```

La libreria DLL creata è alquanto banale ma è più che sufficiente per mostrare come implementare una DLL all'interno del proprio programma. Si apre Visual Studio, si seleziona NUOVO da menu "FILE", in modo che compare a video la seguente maschera:

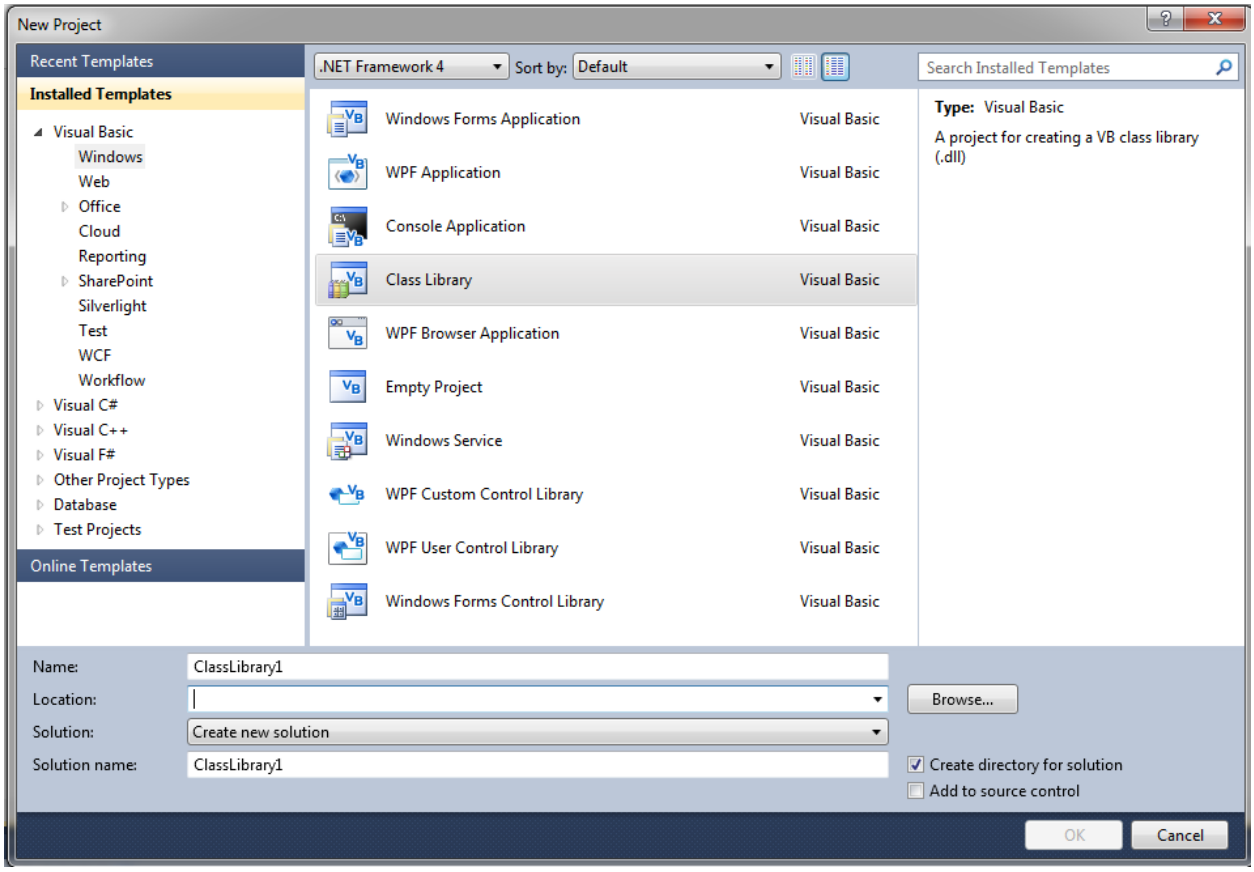

Figura 1

Successivamente si seleziona Visual Basic come linguaggio di programmazione e Class Library come tipo di progetto. Una volta scritto il codice riportato sopra si compila il tutto generando un file con estensione DLL (es: test.dll). A questo punto si apre il nostro programma (scritto per esempio in C#) e si aggiunge la relativa DLL. I passi sono i seguenti:

- Si aggiunge il riferimento alla DLL appena creata:

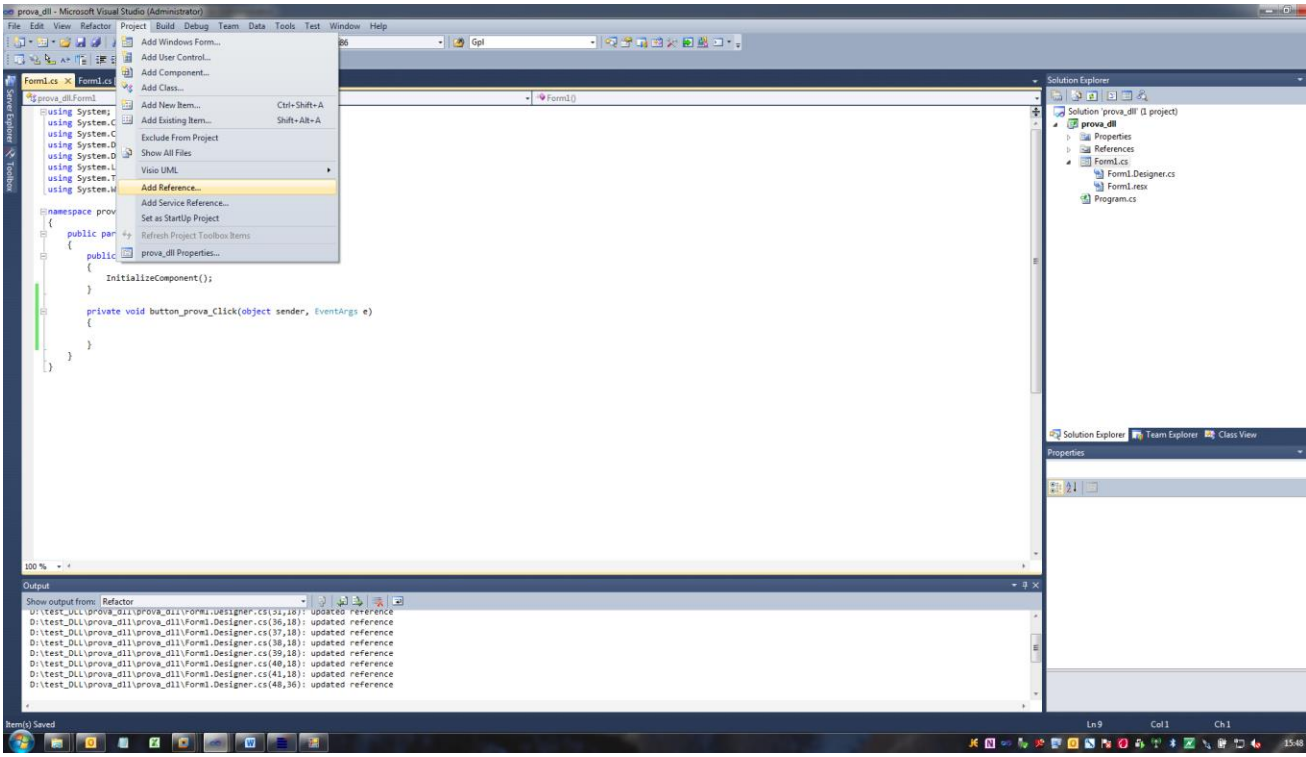

Figura 2

- Si seleziona il file DLL (cliccando eventualmente su BROWSE).

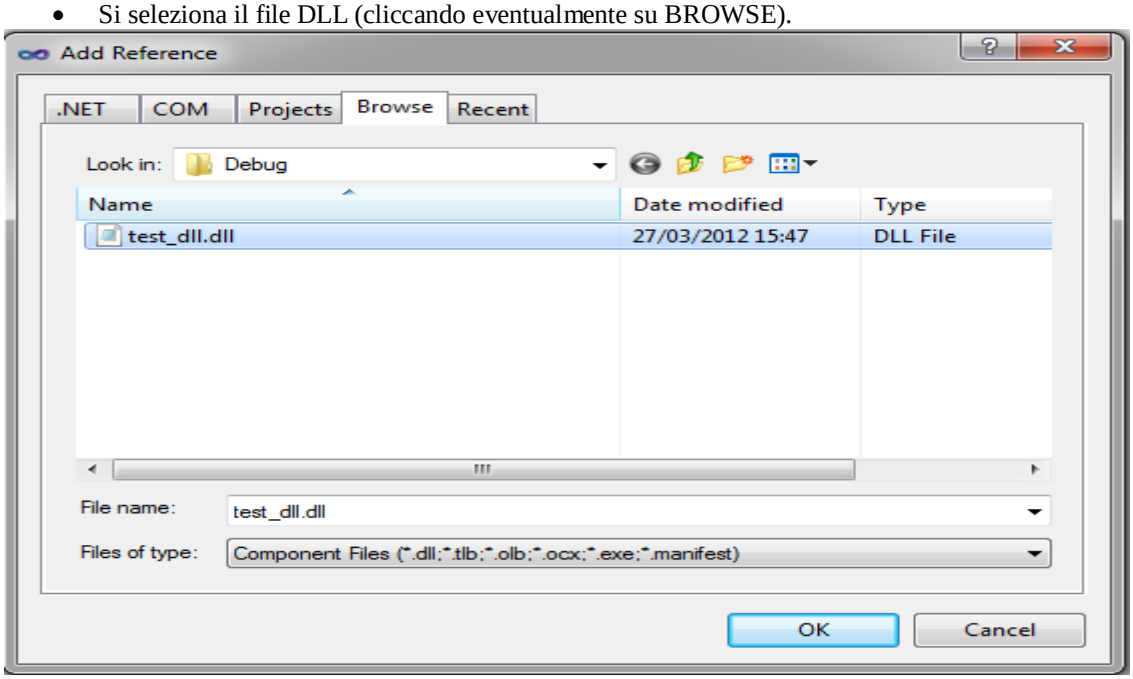

Figura 3

- Inserire la riga seguente: `using test_dll;`

A questo punto la libreria DLL è pronta per l'uso. L'esempio seguente mostra la struttura di base dell'applicativo scritto in C#:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using test_dll;

namespace prova_dll
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button_prova_Click(object sender, EventArgs e)
        {
            test_dll.Class1 t = new test_dll.Class1();
            t.visualizza();
        }
    }
}
```

I vantaggi legati all'uso della DLL sono principalmente di ordine del codice, ossia l'uso della DLL permette di suddividere il codice in parti concettualmente separate. Questo porta ad avere una libreria caricata in memoria che può essere eseguita da più programmi. Questa procedura si chiama **loading on demand** e descrive appunto il nocciolo della questione: Carico in memoria solo le librerie che di volta in volta mi servono. Un grande svantaggio delle DLL sta nel fatto che se viene effettuato un aggiornamento della DLL possono verificarsi i così detti **breaking changes** ossia la DLL può non funzionare correttamente (es: all'interno della DLL vi è una funzione che invece di ritornare NULL mi ritorna un intero).

Pertanto le librerie DLL vengono caricate in memoria quando il processo ne ha bisogno (chiamata di funzione). Le librerie dinamiche vengono caricate nello spazio di memoria del processo, ma per evitare che il codice della DLL e del processo occupino la stessa posizione in memoria, il sistema operativo rimappa ogni riferimento al contenuto della memoria nel codice della DLL.