

GERS

GUIA GENERAL PARA CONSUMIR EL WEB SERVICE DE NEPLAN 360 DESDE PROGRAMACION

Kevin Burbano
M.Sc. Carlo Viggiano

10/7/2018

NEPLAN | OEM / SaS ServicioWeb

Las aplicaciones GIS, SCADA o Smart Grid proporcionadas pueden tener acceso directo a las funciones de NEPLAN 360 si necesidad de uso de un navegador de internet, en particular para todos los módulos de cálculo. Un gran número de servicios Web están incluidos para este propósito, p.e.

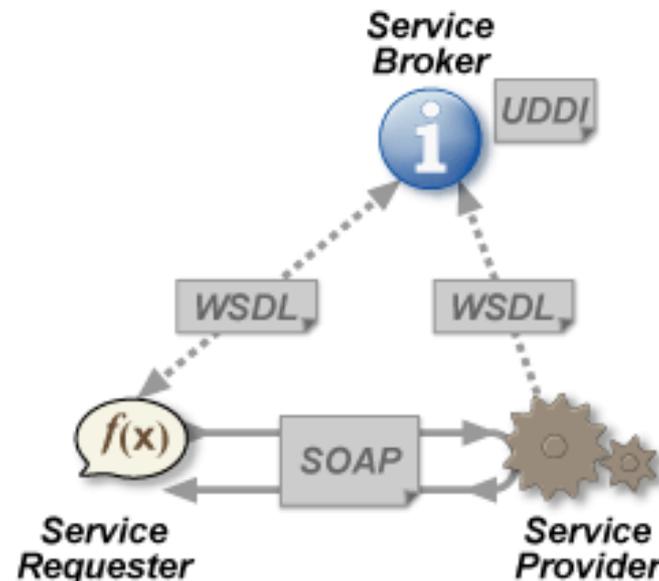
- Para crear y modificar la red (con o sin representación gráfica)
- Para asignar cargas, mediciones o estados de switches
- Para mejorar cualquier cálculo disponible en NEPLAN 360
- Para recuperar todos los resultados de cálculo
- Para administración de usuarios
- Para gestionar errores
- Para monitorear, controlar y optimizar una red

En este caso, el software está alojado en un servidor NEPLAN Internet o Intranet y se puede utilizar sin interfaz gráfica de usuario.

Nota: También existe una API desarrollada en C/C++, que simplifica la integración de NEPLAN en aplicaciones externas. Emplea librerías limitadas para otros desarrollos.

¿Qué es un Web Service?

Un web service es un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como internet.

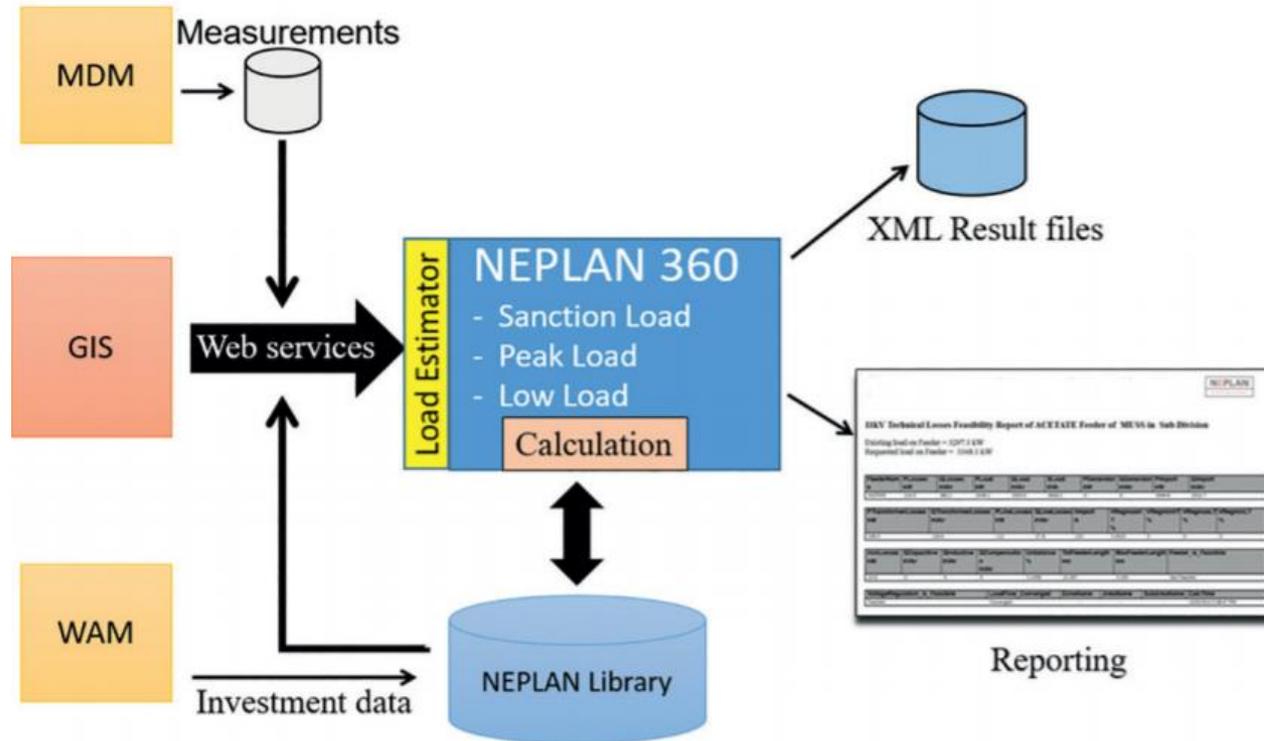


Ventajas de un Web Service

- Permiten que exista interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar y abiertos.
- Emplean estándares y protocolos basados en texto, lo que hace más fácil el acceso a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, los servicios Web pueden tomar ventaja de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

Ventajas de un Web Service

■ Ejemplo



Introducción a la programación orientada a objetos(POO.)

-Programación Orientada a Objetos: es un paradigma de programación donde los objetos manipulan los datos de entrada para la obtención de datos de salida específicos y cada objeto ofrece una funcionalidad especial.

-Clases: es una referencia o “model” del que luego se pueden crear múltiples objetos con características similares.

-Objetos: es una entidad que tienen un determinado "estado" (atributos), "comportamiento" (método) e "identidad".

Introducción a la programación orientada a objetos(POO.)

Ejemplo: Un molde de gelatinas (la clase), y las gelatinas creadas empleando un molde (objetos) con colores y sabores diferentes.

Molde gelatina (la clase)



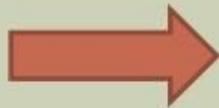
Gelatinas creadas con el molde (objetos)



Introducción a la programación orientada a objetos(POO.)

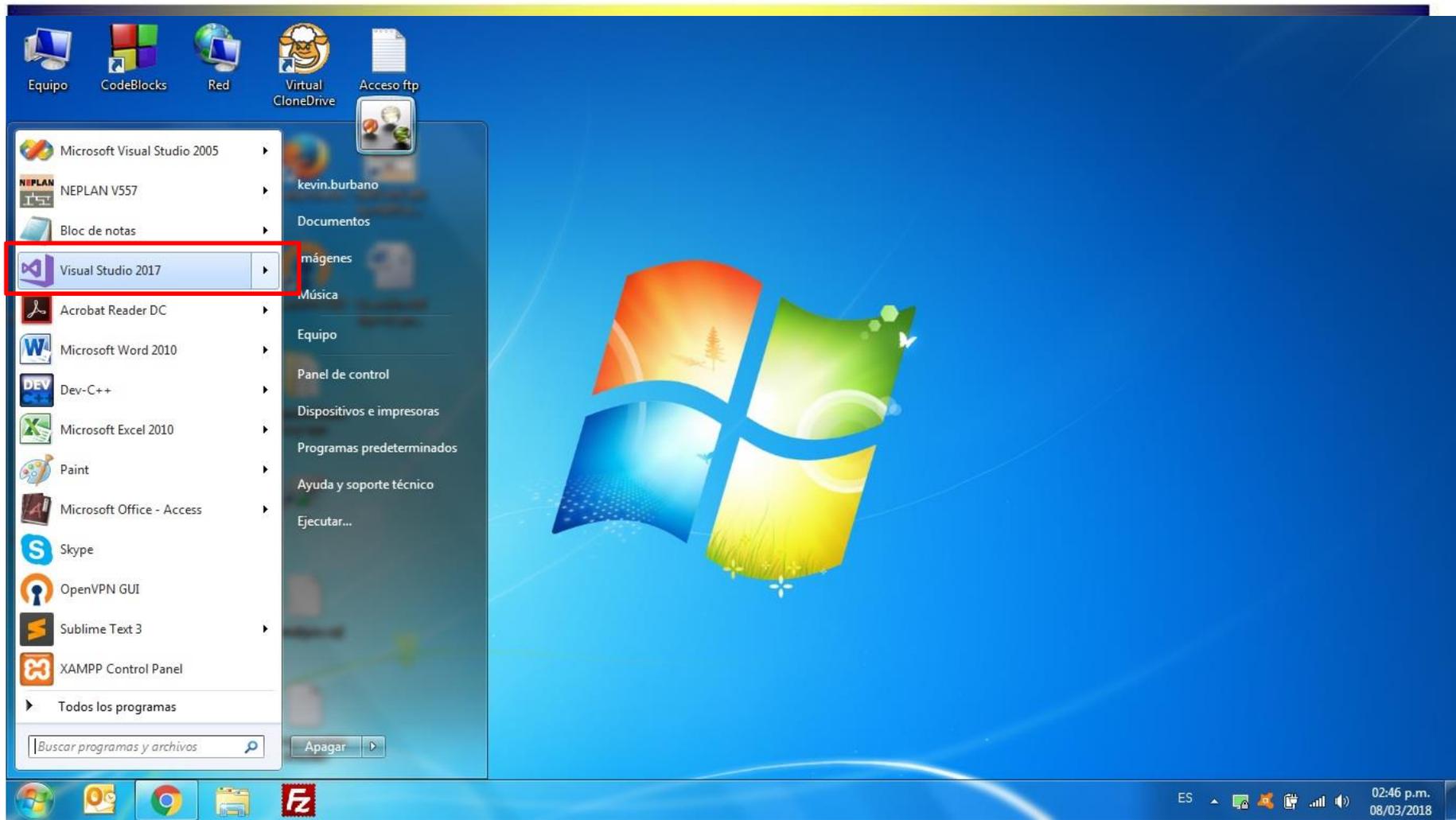
Los atributos son las características, cualidades, propiedades distintivas de cada clase. Contienen información sobre el objeto. Determinan la apariencia, estado y de más particularidades de la clase. Varios objetos de una misma clase tendrán los mismos atributos pero con valores diferentes.

Observe en el ejemplo de las gelatinas, todas las gelatinas (objetos o instancias) creadas a partir del molde (clase) poseen los mismos atributos(color, sabor) pero con valores diferentes (unas son de fresa, otras de limón).

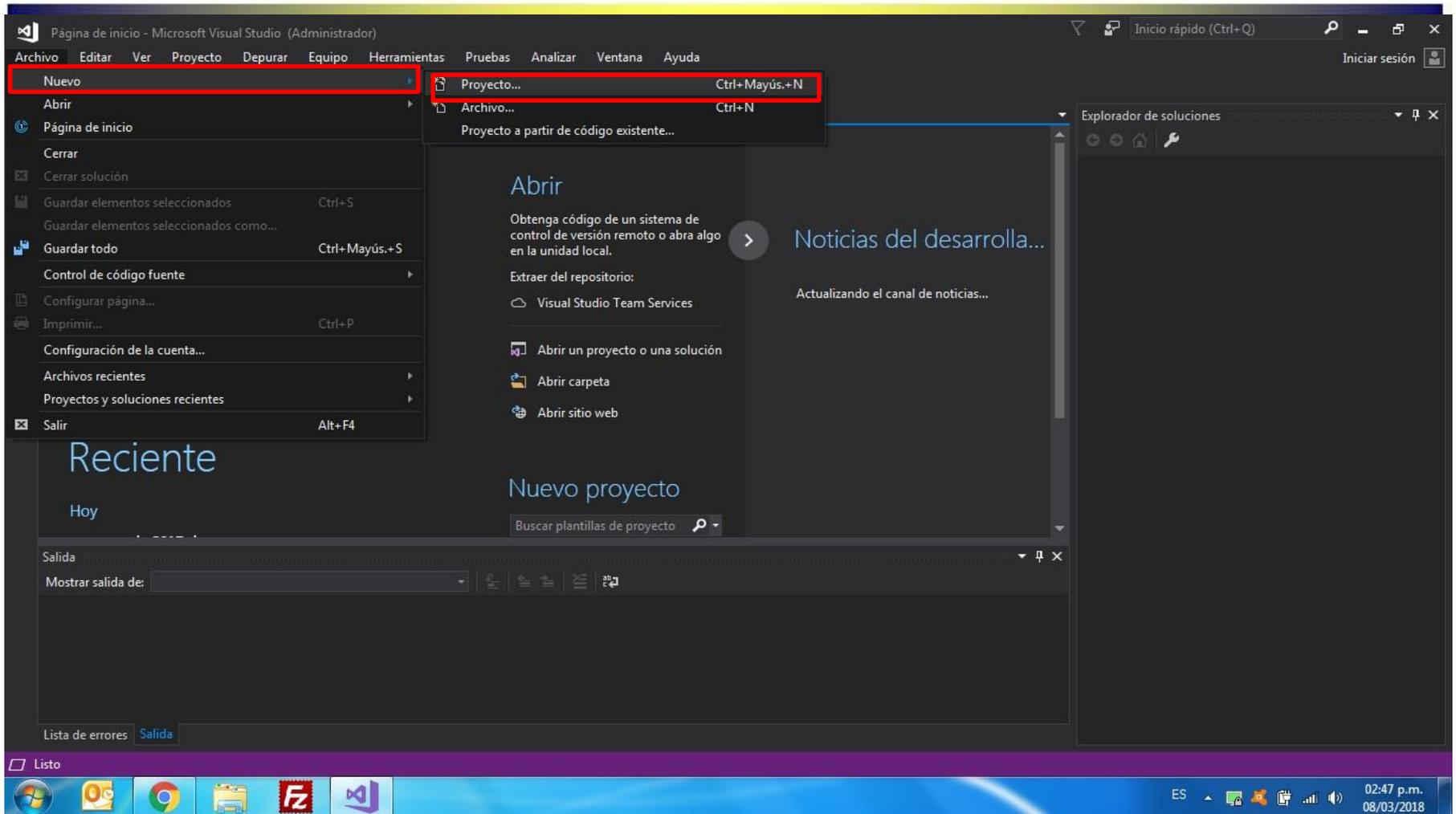


Guía para consumir un web service proporcionado por Neplan 360 desde una aplicación de consola en C#.

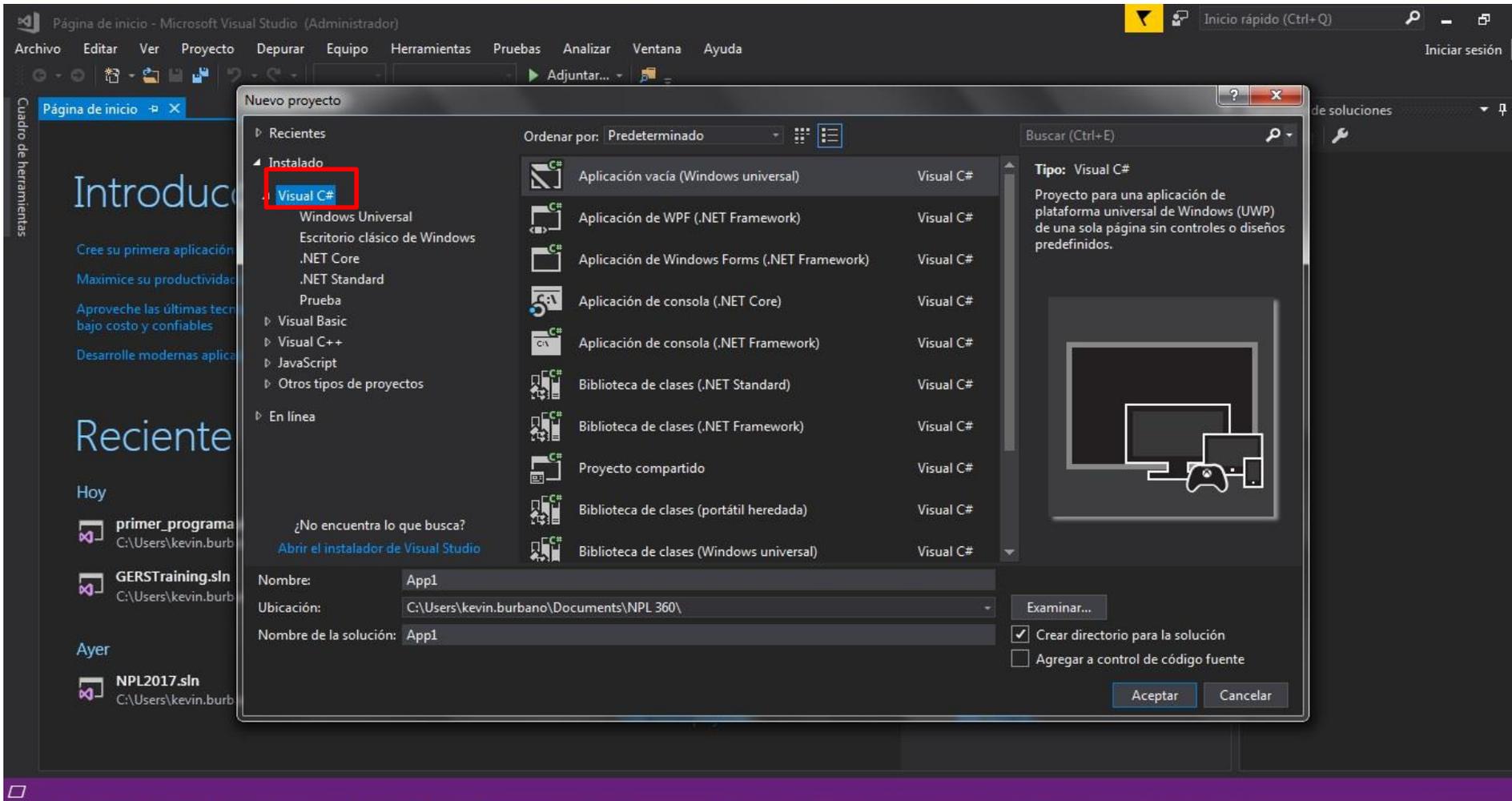
1- Abrir compilador Visual Studio 2017



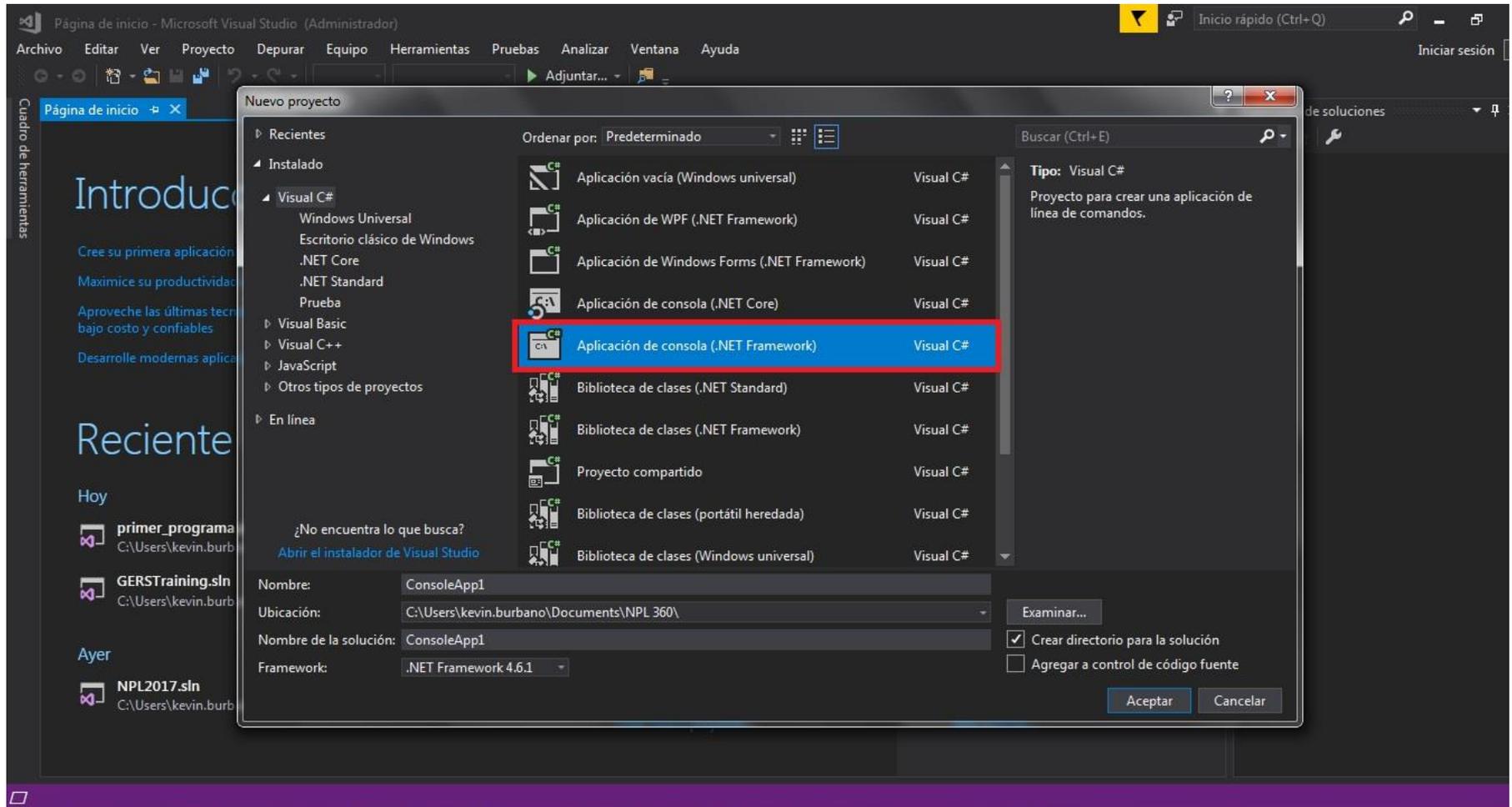
2- Crear nuevo proyecto



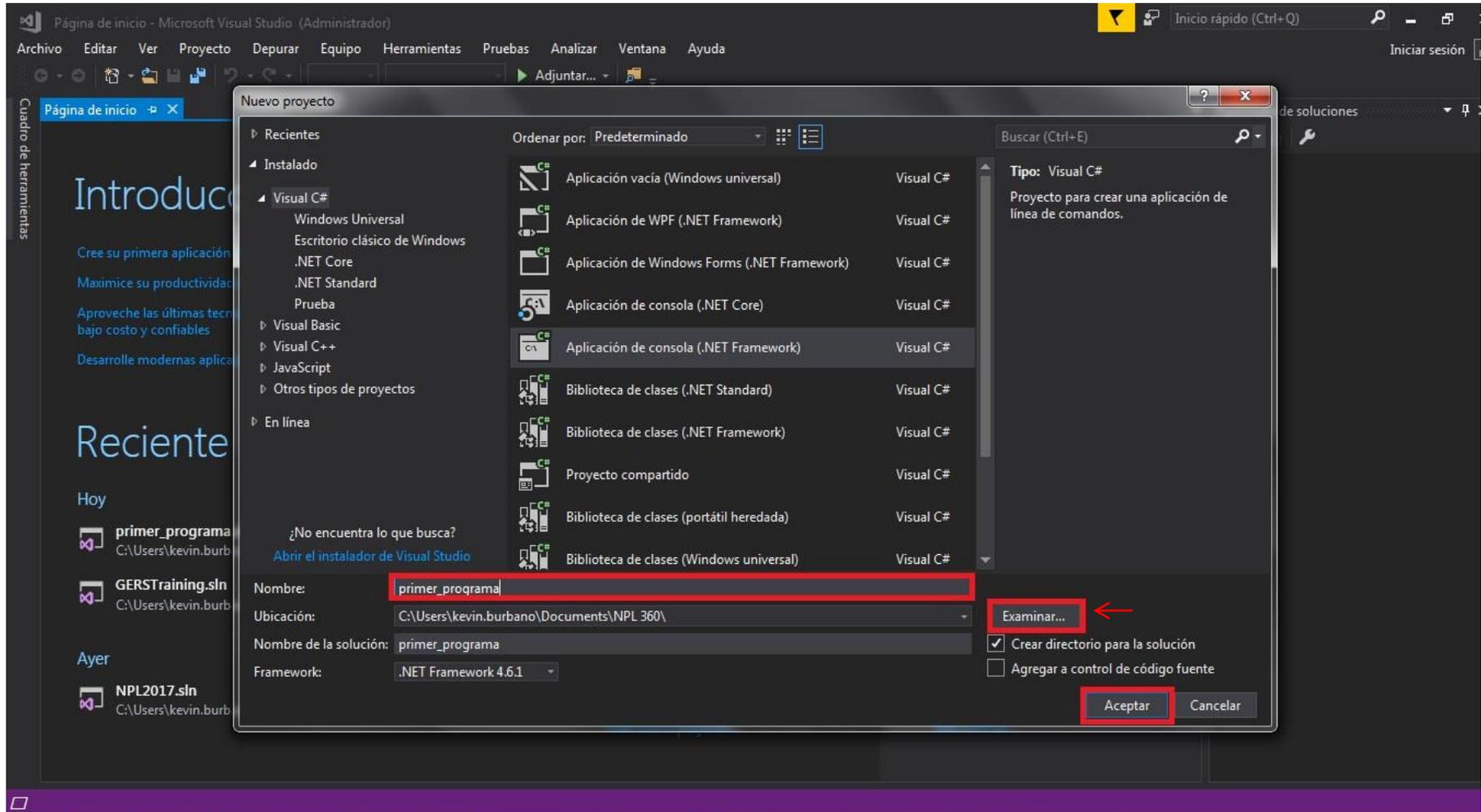
3-Seleccionar Visual C#



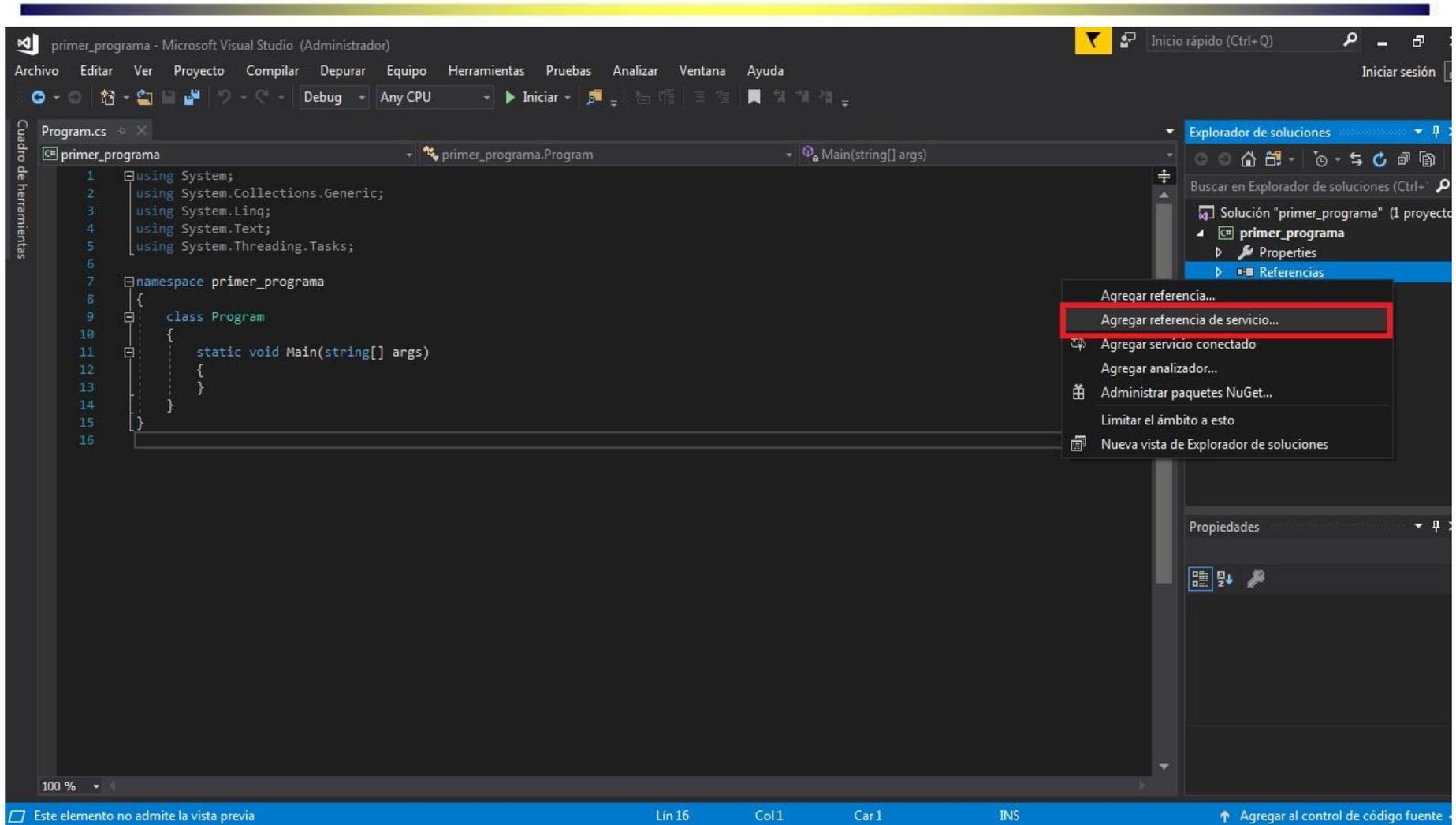
4-Seleccionar aplicación de consola(.NET Framework)



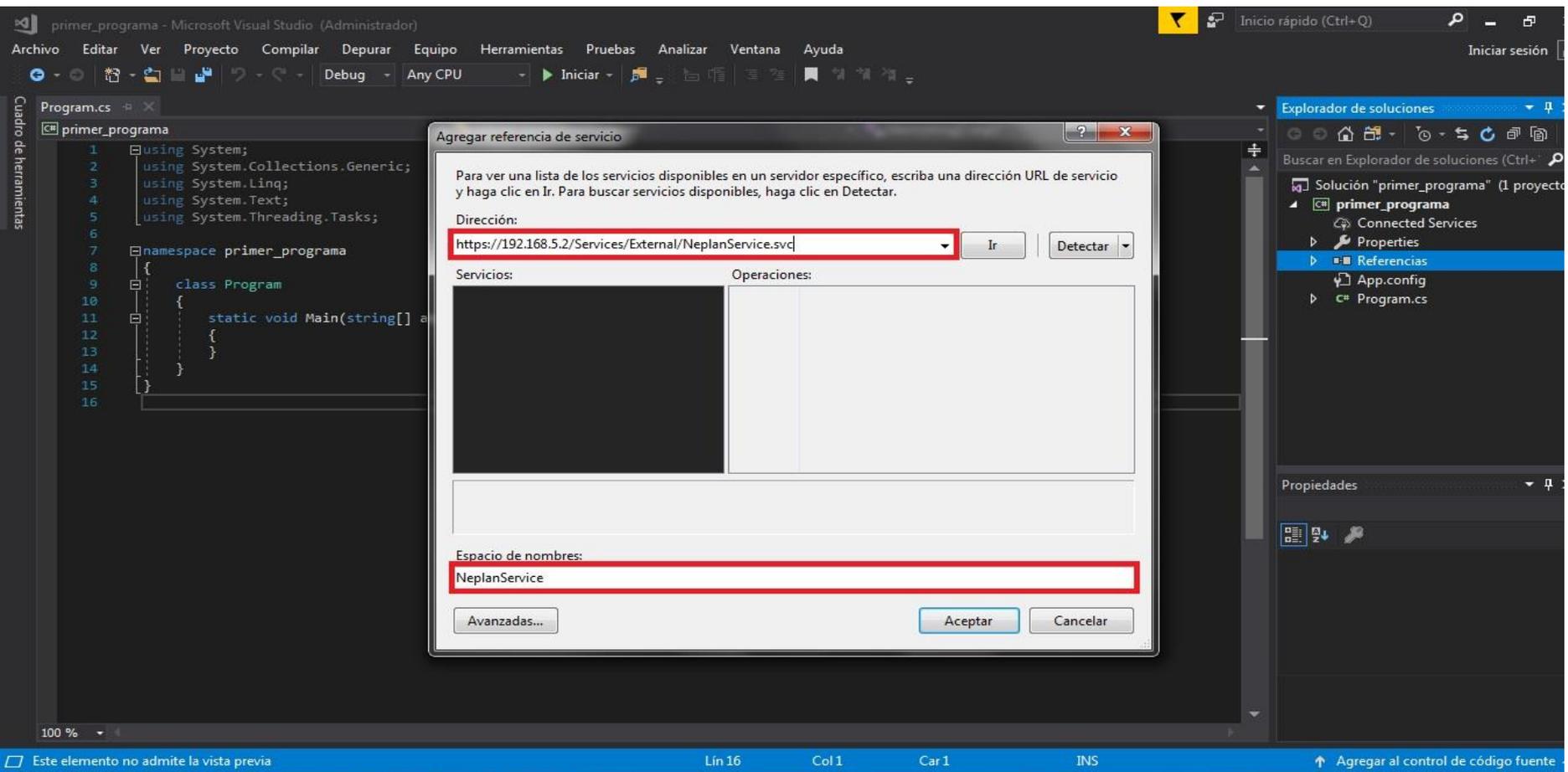
5-Agregar un nombre y la ubicación en donde se guardara el proyecto, después de esto clic en aceptar



6-Clic derecho en referencias, seleccionar “agregar referencia de servicio”.



7-Agregar la dirección en donde se encuentra el WS de Neplan 360, esta puede variar según en donde se encuentren instaladas las licencias.



En espacio de nombres, lo cambiamos a NeplanService.

8-Clic en ir, seleccionar Neplan service, clic en aceptar

The screenshot shows the Visual Studio IDE with a C# project named 'primer_programa'. The code in Program.cs is as follows:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace primer_programa
8 {
9     class Program
10    {
11        static void Main(string[] args)
12        {
13        }
14    }
15 }
16
```

The 'Agregar referencia de servicio' dialog box is open, displaying the following information:

- Dirección:** `https://192.168.5.2/Services/External/NeplanService.svc`
- Servicios:** A list containing one item: `NeplanService`.
- Operaciones:** A section with the text: "Seleccione un contrato de servicio para ver sus operaciones."
- Estado:** "1 servicios encontrados en la dirección 'https://192.168.5.2/Services/External/NeplanService.svc'."
- Espacio de nombres:** `NeplanService`

The 'Ir' button and the 'Aceptar' button are highlighted with red boxes, indicating the next steps in the process.

9-Agregar la librería del WS que se usará en el proyecto, debe anteponerse el nombre del proyecto y un punto, en este caso `primer_programa.NeplanService`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using primer_programa.NeplanService;

namespace primer_programa
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Ejemplo 1(Primer_Programa)

En este ejemplo se usará el caso TestSystem en Neplan 360. Se desarrollará una aplicación de consola en C#, para cambiar dos parámetros de la línea “Line1-3”, correr flujo de carga y mostrar sus resultados en un archivo XML, para este ejercicio debe tenerse en cuenta las siguientes librerías importadas en el proyecto:

- `using System;`
- `using System.Collections.Generic;`
- `using System.Linq;`
- `using System.Text;`
- `using System.Threading.Tasks;`
- `using System.Security.Cryptography;`
- `using System.Xml;`
- `using System.Globalization;`

Ejemplo 1(Primer_Programa)

- Se debe crear una clase donde se especificarán los atributos necesarios para conectar con el WS de Neplan 360.
- Esta clase debe ir por fuera de la clase program que se crea por defecto al crear el proyecto.

Ejemplo 1(Primer_Programa)

//clase donde se especifican los atributos necesarios para conectar el webservice

```
public class Webservice
{
    public NeplanServiceClient nepService;
    public ExternalProject ext; //Entrada debe ir en todos los
desarrollos
    private string username = "Usuario360";
    private string password = "Contraseña";
    public string project = "Proyecto a trabajar";
```

Ejemplo 1(Primer_Programa)

```
public Webservice()  
{
```

```
System.Net.ServicePointManager.ServerCertificateValidationCallback  
= ((sender2, certificate, chain, sslPolicyErrors) => true); //checking  
certificate IIS
```

```
nepService = new NeplanServiceClient(); //instantiates  
the neplaservice
```

```
nepService.ClientCredentials.UserName.UserName = username;  
//give the username
```

```
nepService.ClientCredentials.UserName.Password =  
getMd5Hash(password); //give the password
```

Ejemplo 1(Primer_Programa)

```
try
{
    nepService.Open();                //open the service
    Console.WriteLine("Servicio abierto");
    ext = nepService.GetProject(project, null, null, null); //get
the project
    if (ext != null)
        Console.WriteLine("Proyecto obtenido");
    else
        Console.WriteLine("No se pudo obtener el proyecto");
}
```

Ejemplo 1(Primer_Programa)

catch

```
{  
    Console.WriteLine("No se pudo abrir el servicio");  
}  
}
```

Ejemplo 1(Primer_Programa)

```
public void CloseWebservice()
{
    try
    {
        nepService.Close();    //close the service
        Console.WriteLine("Cerrando servicio");
    }
    catch
    {
        Console.WriteLine("No se pudo cerrar el servicio");
    }
}
}
```

Ejemplo 1(Primer_Programa)

- Ahora se agrega correctamente sus atributos y métodos. Se crea una instancia(objeto) de la clase en el método estático main, el cual se crea por defecto con el proyecto.
- Cabe recalcar que antes de este método estático, se crea una variable booleana para retornar y validar los valores de los cálculos.
- También se crea una instancia de la clase program.

Ejemplo 1(Primer_Programa)

```
bool operacion = true;
```

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

```
{
```

```
    //Creamos el objeto webservice derivado de su clase
```

```
    Webservice webservice = new Webservice();
```

```
    //Creamos el objeto ejemplo derivado de su clase
```

```
    Program ejemplo = new Program();
```

Ejemplo 1(Primer_Programa)

- Una vez declaradas las instancias (objetos) de las clases, se prosigue a establecer los métodos que se ejecutaran.
- Se hace una validación de que el servicio se abre correctamente, se ejecutan los métodos y al terminar cerramos el servicio.

Ejemplo 1(Primer_Programa)

```
if (webservice.nepService != null && webservice.ext != null) //Checks if the  
interface was created
```

```
{  
    ejemplo.change_line(webservice);  
    ejemplo.RunLoadFlow(webservice);           //runs a load flow  
    ejemplo.GetResultsLoadFlow(webservice);    //gets results  
}  
  
webservice.CloseWebservice();  
Console.WriteLine("Press any key to stop...");  
Console.ReadKey();  
}
```

Ejemplo 1(Primer_Programa)

- Los métodos declarados anteriormente no existen
- A continuación, se crearán para así se ejecutarlos desde la anterior declaración. Deben crearse fuera del método estático main.

Ejemplo 1(Primer_Programa)

```
//metodo para cambiar parametros a la linea1-3
internal void change_line(WebService webservice)
{
    if (!operacion)
        return;

    string elementname = "Line1-3"; //nombre del elemento
    string elementtype = "Line"; //tipo del elemento
    string attributename = "C1"; //parametro a cambiar
    string c1 = "0.32547"; //valor del parametro

    string attributename2 = "Length"; // parametro a cambiar
    string longitud = "7"; //valor del parametro
```

Ejemplo 1(Primer_Programa)

//Condional que ejecutamos para asignar los atributos deseados, si se asigno correctamente, mostrara un mensaje de exito, si no: lo contrario

```
if (webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename, c1)) //set the PV Installation production to 40kW
```

```
    Console.WriteLine(string.Format("Output of {0} set to {1}C1", elementname, c1));
```

```
else
```

```
{
```

```
    operacion = false;
```

```
    Console.WriteLine("Could not change output of " + elementname);
```

```
}
```

```
if (webservice.nepService.SetElementAttribute(webservice.ext, elementname, elementtype, attributename2, longitud)) //set the PV Installation production to 40kW
```

```
    Console.WriteLine(string.Format("Output of {0} set to {1}longitud", elementname, longitud));
```

```
else
```

```
{
```

```
    operacion = false;
```

```
    Console.WriteLine("Could not change output of " + elementname);
```

```
}
```

```
}
```

Ejemplo 1(Primer_Programa)

//Metodo para correr flujo de carga

```
internal void RunLoadFlow(WebService webservice)
```

```
{
```

```
    if (!operacion)
```

```
        return;
```

```
    //run load flow. It is possible to define the operational state
```

```
    AnalysisReturnInfo analysis = webservice.nepService.AnalyseVariant(webservice.ext, Guid.NewGuid().ToString(), "LoadFlow", "Default", string.Empty, string.Empty, string.Empty);
```

```
    if (analysis.ReturnInfo == 1 && analysis.HasConverged) //returninfo should be 1 if the calculation was done successfully. HasConverged is a special flag only to be used for Load Flow
```

```
        Console.WriteLine("Flujo de carga ejecutado correctamente!");
```

```
    else
```

```
    {
```

```
        Console.WriteLine("No se pudo correr flujo de carga!");
```

```
        operacion = false;
```

```
    }
```

```
}
```

Ejemplo 1(Primer_Programa)

//Metodo para obtener los resultados de flujo de carga

```
internal void GetResultsLoadFlow(WebService webservice)
```

```
{
```

```
    if (!operacion)
```

```
        return;
```

```
    int networkTypeGroup = 0;           //identifier for network results (other indexes refer to  
area, zone, feeder etc.)
```

```
    string[] networkresults =
```

```
webservice.nepService.GetListResultSummary(webservice.ext, "LoadFlow", new DateTime(),  
networkTypeGroup, null);
```

```
    if (networkresults == null || networkresults.Count() != 1)
```

```
    {
```

```
        operacion = false;
```

```
        Console.WriteLine("Could not get the network results!");
```

```
        return;
```

```
    }
```

GERS

Ejemplo 1(Primer_Programa)

else

```
{  
    string plosses = GetXMLAttribute(networkresults[0], "PLosses");  
    string qlosses = GetXMLAttribute(networkresults[0], "QLosses");  
    Console.WriteLine(string.Format("Network      Active      Losses:      {0:0.000}kW",  
Convert.ToDouble(plosses, CultureInfo.InvariantCulture) * 1000));  
    Console.WriteLine(string.Format("Network      Reactive      Losses:      {0:0.000}kVar",  
Convert.ToDouble(qlosses, CultureInfo.InvariantCulture) * 1000));  
}
```

```
int portNumber = 0;      //0 is the first port connection of an element in Neplan
```

```
string elementname = "Line1-3";
```

```
string elementtype = "Line";
```

```
//gets the element results for the external grid
```

```
string result = webservice.nepService.GetResultElementByName(webservice.ext, elementname,  
elementtype, portNumber, "LoadFlow", new DateTime());
```

Ejemplo 1(Primer_Programa)

```
if (result == null)
{
    operacion = false;
    Console.WriteLine("No se pudo obtener los resultados de ");
    return;
}
else
{
    string power = GetXMLAttribute(result, "P"); //gets the power import for the
network
    Console.WriteLine(string.Format("The network is supplied with: {0:0.00} kW", -
Convert.ToDouble(power, CultureInfo.InvariantCulture) * 1000));
}
}
```

Ejemplo 1(Primer_Programa)

//Metodo para encapsular los resultados en archivos XML

```
private string GetXMLAttribute(string result, string xmlelement)
{
    if (string.IsNullOrEmpty(result))
        return null;

    XmlDocument xmldoc = new XmlDocument();
    xmldoc.LoadXml(result);
    XmlNodeList nodeList = xmldoc.GetElementsByTagName(xmlelement);
    if (nodeList == null || nodeList.Count != 1)
    {
        operacion = false;
        return null;
    }
    else

        return nodeList[0].InnerText;
}
}
```

Ejemplo 1(Primer_Programa)

- Para visualizar los resultados en el archivo XML debe hacerse uso de puntos de interrupción en el código.
- Para colocar un punto de interrupción, se usa F9 estando situado en la línea de código a la cual se desea agregar.
- En este caso, se adiciona en el condicional donde se valida el resultado (en el método de obtener resultado de flujo de carga).

Ejemplo 1(Primer_Programa)

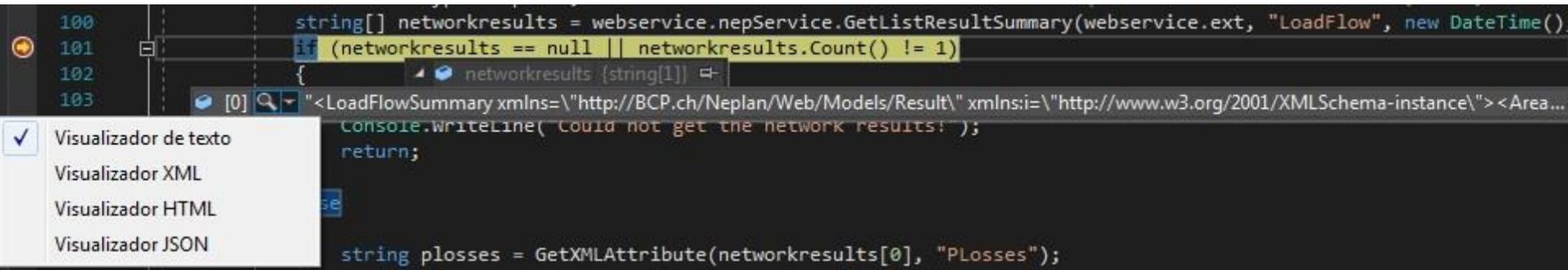
The screenshot displays the Microsoft Visual Studio IDE with a C# program named `Program.cs` open. The code is a method `GetResultsLoadFlow` that interacts with a `WebService` to retrieve network results and calculate losses. The code is as follows:

```
93 //Metodo para obtener los resultados de flujo de carga
94 internal void GetResultsLoadFlow(WebService webservice)
95 {
96     if (!operacion)
97         return;
98
99     int networkTypeGroup = 0; //identifier for network results (other indexes refer to area, zone, feeder etc.)
100     string[] networkresults = webservice.nepService.GetListResultSummary(webservice.ext, "LoadFlow", new DateTime(), networkTypeGroup);
101     if (networkresults == null || networkresults.Count() != 1)
102     {
103         operacion = false;
104         Console.WriteLine("Could not get the network results!");
105         return;
106     }
107     else
108     {
109         string plosses = GetXmlAttribute(networkresults[0], "Plosses");
110         string qlosses = GetXmlAttribute(networkresults[0], "Qlosses");
111         Console.WriteLine(string.Format("Network Active Losses: {0:0.000}kW", Convert.ToDouble(plosses, CultureInfo.InvariantCulture)));
112         Console.WriteLine(string.Format("Network Reactive Losses: {0:0.000}kVar", Convert.ToDouble(qlosses, CultureInfo.InvariantCulture)));
113     }
114
115     int portNumber = 0; //0 is the first port connection of an element in Neplan
116     string elementname = "Line1-3";
117     string elementtype = "Line";
118     //gets the element results for the external grid
119     string result = webservice.nepService.GetResultElementByName(webservice.ext, elementname, elementtype, portNumber, "LoadFlow");
120     if (result == null)
121     {
122         operacion = false;
123         Console.WriteLine("No se pudo obtener los resultados de ");
124         return;
125     }
126 }
```

The IDE interface includes a menu bar (Archivo, Editar, Ver, Proyecto, Compilar, Depurar, Equipo, Herramientas, Pruebas, Analizar, Ventana, Ayuda), a toolbar with icons for file operations and debugging, and a status bar at the bottom showing "Listo", "Lin 122", "Col 17", "Car 17", "INS", and "Agregar al control de código fuente". The right sidebar shows the "Explorador de soluciones" (Solution Explorer) with a project named "Solución 'primer_programa' (1 proyecto)" containing files like "App.config", "Program.cs", and "Properties".

Ejemplo 1(Primer_Programa)

- Al iniciar el código, cuando se pause en el punto de interrupción, se debe posicionar el mouse en networkresults y desplegar la opción, de la siguiente manera.



The screenshot shows a Visual Studio IDE with a code editor and a debugger. The code is in C# and includes a breakpoint at line 101. The variable 'networkresults' is highlighted in the debugger's watch window. A context menu is open over the variable, showing options for different visualizers: 'Visualizador de texto' (selected), 'Visualizador XML', 'Visualizador HTML', and 'Visualizador JSON'. The code in the background includes a call to 'GetListResultSummary' and a check for 'networkresults'.

```
100 string[] networkresults = webservice.nepService.GetListResultSummary(webservice.ext, "LoadFlow", new DateTime());
101 if (networkresults == null || networkresults.Count() != 1)
102 {
103     networkresults {string[1]}
    [0] "<LoadFlowSummary xmlns=\\"http://BCP.ch/Neplan/Web/Models/Result\\" xmlns:i=\\"http://www.w3.org/2001/XMLSchema-instance\\"><Area...
    Console.WriteLine( "Could not get the network results! ");
    return;
    string plosses = GetXMLAttribute(networkresults[0], "Plosses");
```

Se debe seleccionar visualizador XML y nos desplegara la siguiente ventana con los resultados del flujo de carga.

Ejemplo 1(Primer_Programa)

- Parámetros que se asignaron desde la aplicación de consola en C#

The screenshot displays the NEPLAN 360 software interface. The main window shows the configuration for a line section named "LINE1-3". The "Length.. m" field is set to 7000, and the "C(1) .. μF/km" field is set to 0.32547, both of which are highlighted with red boxes. Other parameters include R(1) = 3.93576, X(1) = 19.678801, B(1) = 122.699299, and G(1) = 0. The interface also shows a sidebar with navigation options and a status bar at the bottom.

Parameter	Value	Units
Name	LINE1-3	
Length.. m	7000	Ω/km
Number of lines	1	
R(1) .. Ω/km	3.93576	
X(1) .. Ω/km	19.678801	
C(1) .. μF/km	0.32547	
B(1) .. μS/km	122.699299	
G(1) .. μS/km	0	
Z LN .. Ω/km	0	
Z NN .. Ω/km	0	
Ir max (low) .. A	0	
Ir max (med) .. A	0	
Ir max (high) .. A	0	
Ir max (def) .. A	0	
R(0) .. Ω/km	0	
X(0) .. Ω/km	0	
C(0) .. μF/km	0	
B(0) .. μS/km	0	
Z LG .. Ω/km	0	
Z GG .. Ω/km	0	
Ier max .. A	0	
Reduction factor	1	
Type of line	Overhead	
Asymmetrical PI model	<input type="checkbox"/>	

Ejemplo 2(Listar_elementos)

En este ejemplo se usara el caso TestSystem en Neplan 360,se desarrollara una aplicación de consola en C#, para crear una lista de elementos del mismo tipo, cambiar el factor de escalamiento p a todos los elementos que estarán en esta lista, en este caso cargas.

Se hará uso de las misma librerías que se utilizaron en el ejercicio anterior.

Recordar realizar los pasos que se explican en las diapositivas: 21,27,29,31

Ejemplo 2(Listar_elementos)

En este ejemplo se usará el caso TestSystem en Neplan 360 y se desarrollará una aplicación de consola en C# para crear una lista de elementos del mismo tipo, cambiar el factor de escalamiento p a todos los elementos que estarán en esta lista (solo cargas en este caso).

Se hará uso de las misma librerías que se utilizaron en el ejercicio anterior.

Recordar realizar los pasos que se explican en las diapositivas: 21,27,29,31

Ejemplo 2(Listar_elementos)

Creamos el metodo para listar elementos.

```
internal void listar_elementos(Webservice webservice)
{
    if (!operacion)
        return;

    //Creamos una variable que contendra el objeto webservice
    junto con la variable nepservice
        var neplan = webservice.nepService;

    //Creamos un diccionario donde se almacenara el nombre de
    cada elemento
        var elementNames = new Dictionary<string, string>();
```

Ejemplo 2(Listar_elementos)

```
//Creamos un diccionario donde se guardara el id y tipo de cada elemento
```

```
var elementTypes = new Dictionary<string, string>();  
//first string is Id second string is type
```

```
//Usamos la variable neplan para llamar el metodo  
GetAllElementsOfElementType que es el encargado de obtener una  
lista de elementos
```

```
neplan.GetAllElementsOfElementType(webService.ext, "Load", ref  
elementNames, ref elementTypes);
```

```
//Creamos esta variable donde guardaremos el valor que queremos  
asignarle al factor de escalamiento p
```

```
string pScalingFactor = "0.3";
```

Ejemplo 2(Listar_elementos)

```
//Hacer el cambio en la lista por nombre de elementos
foreach (var item in elementNames) //iterate over the list
of elements
    {
        var elementName = item.Value;
        if
(!neplan.SetElementAttribute(webbservice.ext, elementName,
"Load", "ScalingFactorElementP", pScalingFactor))
            Console.WriteLine($"No se pudo cambiar
el valor de la carga {elementName}");
        else
            Console.WriteLine($"Cambios aplicados
correctamente en : {elementName}");
    }
}
```