التعامل مع ال TreeView في السي شارب

1. قم باضافة مكون من النوع Treeview من الى النموذج (Form1)
2. انقر على السهم الموجود في الزاوية العليا اليمنى من المكونtreeView1 انظر الشكل (Figure1)
3. ستظهر لك نافذة باسم TreeView Tasks (انظر الشكل Figure2)

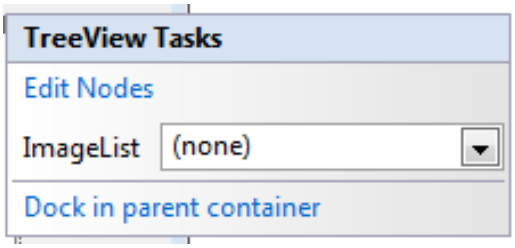


Figure1

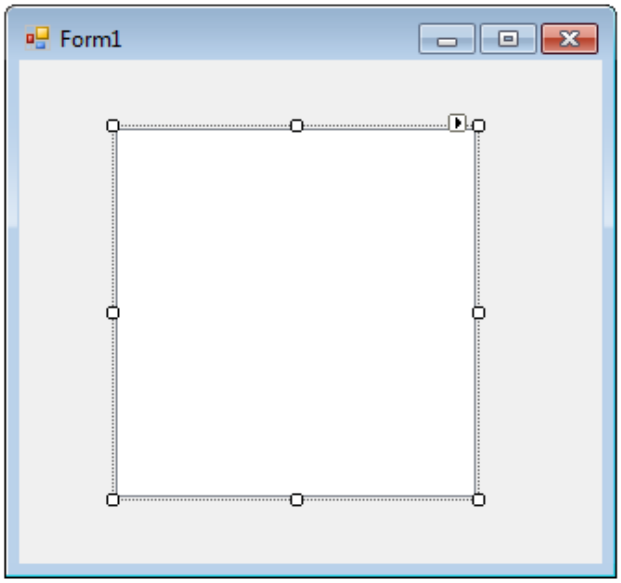


Figure2

4. بعد ذلك نقر على العبارة Edit node ستظهر لك نافذة باسم TreeNode Editor انظر الشكل (Figure3)
5. لكي تضيف جذورا انقر على الزر (Add root)
توضيح نطلق جذر على العقدة التي لا تملك أي اب أي تلك التي تقع عند بداية الشجرة
6. اذا اردت اضافة ابناء انقر على الزر (Add Child).
طبعا لا يجوز اضافة ابناء دون ضافة أي اب لذا ترى ترى الزر (Add Child) غير فعال مالم لم تختر عقدة موجودة او تنشيء عقدة .
انظر الشكل (Figure 4)

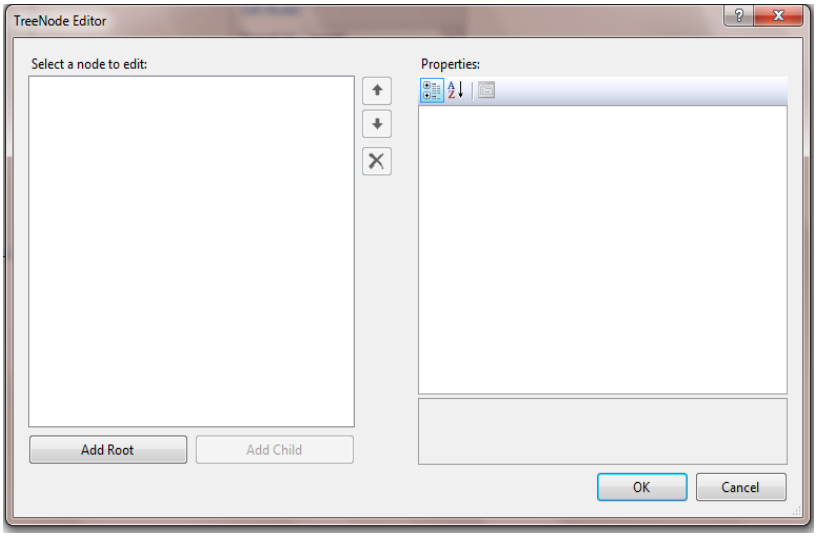


Figure3

لاحظ ظهور عقد الشجرة على يسار الشكل (Figure 4) ولاحظ ظهور خواص كل عقدة على يمين الشكل . لاحظ ان الخواص مرتبة في مجموعات منطقية متشابهة (مجموعة مختصة بالمظهر ومجموعة بالاداء..الخ) طبعا كل عقدة هي Object من الصنف TreeNode Class

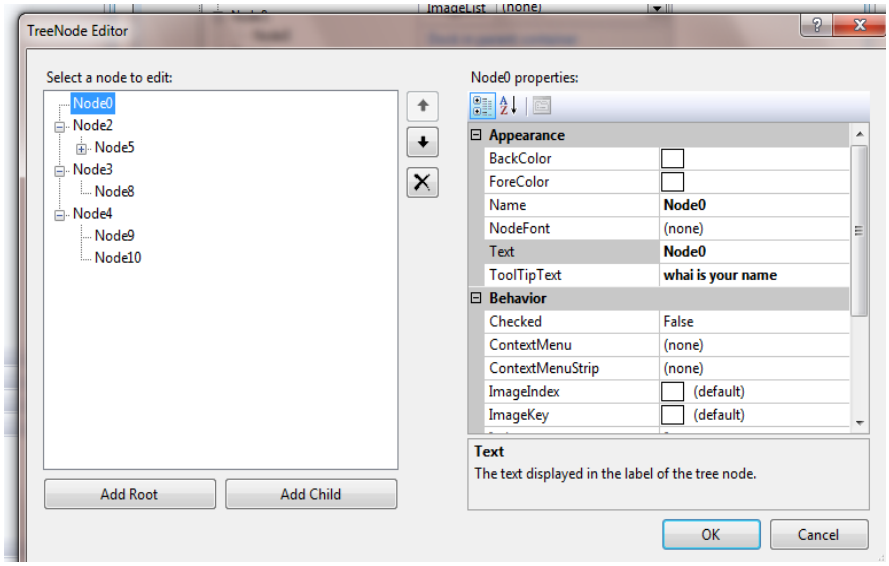


Figure4

لكي نمنح لكل عقدة ايقونة معينة نقوم بالاتي:

I. نقوم بإضافة مكون من النوع ImageList كما تعلمنا ذلك سابقا واشحنه بالايقونات
II. نذهب الى الخاصية ImageList من المكون treeView1 ونربطها مع المكون imageList1
III. انقر على السهم الموجود في الزاوية العليا اليمنى من المكون treeView1 ستظهر لك النافذة التي في الشكل (Figure 4)
IV. اختر عقدة معينة من الشجرة على يسار النافذة
V. من المجموعة Behavior اذهب الخاصية ImageIndex واختر رقم الايقونة التي ستكون متربطة بالعقدة الحالية عندما تكون العقدة في حالتها الطبيعية أي بدون اختيار
VI. اذهب الى الخاصية SelectedImageIndex لاختيار رقم الايقونة عندما الايقونة في حالة اختيار (اذا لم تقم بتغيير هذه الخاصية فان قيمتها ستكون نفس قيمة ImageIndex)
VII. الخاصيتان ImagaeIndexKey و SelectedImageKey تقومان بنفس عمل الخاصيتين السابقتين ولكن قيمهما اسماء الايقونات وليس ارقامها في قائمة الايقونات

نستطيع تغيير النص الذي يخص كل عقدة من الخاصية text كما اننا نستطيع تغيير التنسيق من الخاصية NodeFont ولون الخلفية من الخاصية BackColor (ننبه مرة اخرى الى ان كل الخواص السابقة هي للصنف TreeNode)

والان نستطيع ان نفعل برمجيا ما فعلناه ما فعلنا سابقا بشكل يدوي

افتح مشروعا جديدا كالتالي :

File>New>New Project > Windows Application

- اضف مكون من النوع TreeView الى النموذج (الى الفورم)
- اضف مكون من النوع Button واجعل الخاصية text لهذا المكون تساوي "Add Childs"
- انقر على هذا الزر مرتين ثم اكتب الكود التالي:

```
private void button2_Click(object sender, EventArgs e)
   {
     treeView1.BeginUpdate();
     treeView1.Nodes.Clear();
     treeView1.Nodes.Add("Parent1");
     treeView1.Nodes[0].Nodes.Add("Child 1");
     treeView1.Nodes[0].Nodes.Add("Child 2");
     treeView1.Nodes[0].Nodes[1].Nodes.Add("Grandchild");
     treeView1.Nodes[0].Nodes[1].Nodes[0].Nodes.Add("Great Grandchild");
     treeView1.EndUpdate();
    }
```

يقوم السطر الاول بايقاف عملية اعادة رسم المكون treeView1 وذلك لان عملية رسم المكونات مستمرة كلما حدث تغيير لشكل

المكون يمكننا الاستغناء عن هذا السطر ولكن هذا يرفع فعالية البرنامج وقل نفس الشيء عن السطر الاخير الذي يعيد تفعيل عملية اعادة الرسم للمكون.

السطر الثاني

```
treeView1.Nodes.Clear();
```

يقوم هذا السطر بمسح كل الجذور من الشجرة وتبعا لذلك بمسح جميع ابناء الجذور الممسوحة لاحظ ان استخدام المنهج Clear يدل على ان الحقل Nodes هو Collection وبناء على هذا يمكن استخدام المناهج Add, Remove الخ

بعد تنفيذ هذا السطر تصبح الشجر خالية تماما .

```
     treeView1.Nodes.Add("Parent1");
```

يقوم هذا الجزء باضافة اول جذر للشجرة

```
treeView1.Nodes[0].Nodes.Add("Child 1");
treeView1.Nodes[0].Nodes.Add("Child 2");
```

يقوم هذا الجزء باضافة ابناء للجذر الاول

```
treeView1.Nodes[0].Nodes[1].Nodes.Add("Grandchild");
```

اما هذا السطر فهو يضيف ابنا الى **الابن الثاني للجذر والان انظر الى الشجرة بعد تشغيل البرنامج**

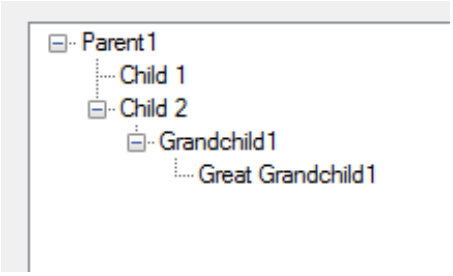


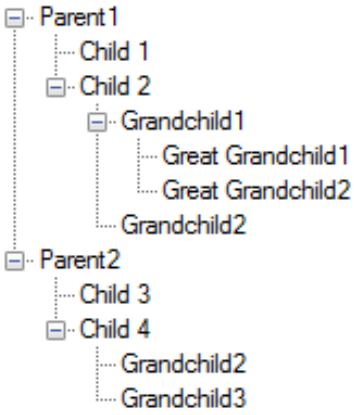
Figure 5

قارن هذه الشجرة بالشجرة الناتجة من الكود التالي

```
private void button2_Click(object sender, EventArgs e)
    {
       treeView1.BeginUpdate();
       treeView1.Nodes.Clear();
       treeView1.Nodes.Add("Parent1");
       treeView1.Nodes[0].Nodes.Add("Child 1");
       treeView1.Nodes[0].Nodes.Add("Child 2");
       treeView1.Nodes[0].Nodes[1].Nodes.Add("Grandchild1");
       treeView1.Nodes[0].Nodes[1].Nodes.Add("Grandchild2");
       treeView1.Nodes[0].Nodes[1].Nodes[0].Nodes.Add("Great Grandchild1");
       treeView1.Nodes[0].Nodes[1].Nodes[0].Nodes.Add("Great Grandchild2");

        treeView1.Nodes.Add("Parent2");
        treeView1.Nodes[1].Nodes.Add("Child 3");
        treeView1.Nodes[1].Nodes.Add("Child 4");
        treeView1.Nodes[1].Nodes[1].Nodes.Add("Grandchild2");
        treeView1.Nodes[1].Nodes[1].Nodes.Add("Grandchild3");

        treeView1.ExpandAll();
        treeView1.EndUpdate();
        }
```



لاحظ الفرق بين بناء الشجرتين .ايضا لاحظ السطر قبل الاخير استخدام المنهج `ExpandAll` وذلك لتوسعة كل العقد واظهار كل الابناء كما اننا اضفنا جذرا اخر بالسطر

```
treeView1.Nodes.Add("Parent2");
```

Figure 6

الكود التالي يمر عبر كل عقدة من عقدة الشجرة حسب البحث بالعمق (Depth-first search) وذلك لتغيير النص الموجود في كل عقدة:

```
//=========================================================================
        void MoveViaAll(TreeNode n)
        {
            foreach (TreeNode child in n.Nodes)
            {
                index++;
                child.Text = "node"+index.ToString();
                MoveViaAll(child);

            }
        }

//=========================================================================
        private void button3_Click(object sender, EventArgs e)
        {
            int i;
            if(treeView1.Nodes.Count>0)
                for ( i = 0; i < treeView1.Nodes.Count ; i++)
                {
                    index++;
                    treeView1.Nodes[i].Text = "node" + index.ToString();
                    MoveViaAll(treeView1.Nodes[i]);
                }
        }
//=========================================================================
```

تحتوي الدالة MoveViaAll على استدعاء ذاتي وذلك للمرور عبر كل الابناء وكذلك ابناء الابناء و... الخ من الاحفاد

أما الحلقة الموجودة فهي لضمان دخول الجذور ومن ثم ابناء الجذور عبر الدالة MoveViaAll.

البرنامج التالي يوضح المرور عبر كل عقدة من عقدة الشجرة البحث حسب Breadth-first search

```
void AddChildrenList(ArrayList List)
        {
            TreeNode n;
            while (List.Count > 0)
            {
                n = (TreeNode)List[0];
                foreach (TreeNode child in n.Nodes)
                {
                    index++;
                    child.Text = "node" + index.ToString();
                    List.Add(child);
                }
                List.RemoveAt(0);
            }

        }
        private void button4_Click(object sender, EventArgs e)
        {
            int i;
            index = 0;
            ArrayList List = new ArrayList();
            if (treeView1.Nodes.Count > 0)
                for (i = 0; i < treeView1.Nodes.Count; i++)
                {
```

```
                index++;
                treeView1.Nodes[i].Text = "node" + index.ToString();
                List.Add(treeView1.Nodes[i]);
            }
        AddChildrenList(List);

    }
```

لاحظ البحث نتائج المرور بعمق (Figure 7) والمرور بالعرض(Figure 8)

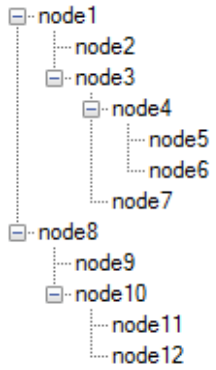


Figure 7

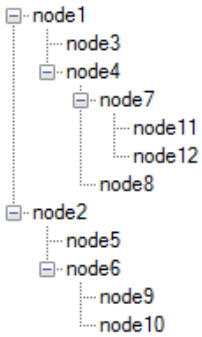


Figure8

لاحظ وجود استدعاء ذاتي في المرور بالعمق بينما نجد طابور يخزن العقد في المرور بالعرض

```
List.Add(child);

MoveViaAll(child);
```

ولاحظ كذلك اننا نحذف العقدة الاولى من الطابور بعد اضافة ابناءها وهذا تاكيد لمبدأFIFO

```
 List.RemoveAt(0);
```

طبعا البحث بالعمق ايضا امر مهم جد ا فهو يساعدنا على نقل العقد بمحتوياتها بين شجرتين انظر الكود التالي:

```
private void CloneChild(TreeNode OldNode, TreeNode NewNode)
        {
            foreach (TreeNode Child in NewNode.Nodes)
            {
                TreeNode n = new TreeNode(Child.Text);
                OldNode.Nodes.Add(n);
                CloneChild(n, Child);
            }

        }
        private void button5_Click(object sender, EventArgs e)
        {
            TreeNode n1 = treeView1.SelectedNode;
            TreeNode n2 = treeView2.SelectedNode;
            TreeNode n3 = new TreeNode(n2.Text);
            n1.Nodes.Add(n3);
            CloneChild(n3,n2);
        }
```

الكود التالي يوضح الطريقة التي تجعلنا نعرف مستوى العقدة (معروف ان مستوى الجذور هو صفر وان ابناءها في المستوى1 وابناء الابناء في المستوى 2 والخ)

```
public int NodeLevel(TreeNode node)
        {
            int level = 0;
            node = node.Parent
            while (node != null)
            {
                level++;
                node = node.Parent
            }
            return level;
        }
```

يفيدنا أيضا هذا الكود في معرفة الطريق من عقدة معينة إلى الجذر الذي تتبعه هذه العقدة (كيف؟ متروك كتمرين).

نستطيع ان نغير الايقونة التي تملكها كل عقدة وذلك بواسطة قيمة الخاصية ImageIndex

```
private void button1_Click(object sender, EventArgs e)
        {
            treeView1.SelectedNode.ImageIndex = 10;
            treeView1.SelectedNode.SelectedImageIndex = 10;
        }
```

طبعا ان افترض ان لدي مكون من النوع imageList مشحون بالايقونات اذا لم يكن الامر فاضف مكون من النوع imageList

واشحنه بالصور كما تعلمنا آنفا. ثم نفذ الكود التالي

```
private void button6_Click(object sender, EventArgs e)
        {
            treeView1.ImageList = imageList1;
            treeView1.SelectedNode.ImageIndex = 4;
            treeView1.SelectedNode.SelectedImageIndex = 4;
        }
```

لاحظ اننا في الشفرة الاخيرة غيرت قيمة ImageIndex وهذا يعني انني اخترت الايقونة الخامسة (لماذا الخامسة وليس الرابعة ؟) بدلا من الايقونة الحادية عشرة ولاضير في ذلك طالما ان القيمة الممنوحة لImageIndex واقعة بين القيمة 0 و والقيمة imageList1.Images.Count-1 .

لاحظ بعد تنفيذ الشفرة أن معظم عقد الشجرة تاخذ الايقونة الاولى وذلك لان القيمة الافتراضية للخاصيةImageIndex هي صفر لاي عقدة مالم نقل غير ذلك صراحة....

سنرى كيف نقوم بعملية نقل بين عقدة مع ابناءها بين شجرتين بواسطة السحب والإفلات بين شجرتين

1. قم بفتح مشروع جديد بواسطة

File>New>New Project > Windows Application

2. اضف مكونين من النوع TreeView وليكن الاول منهما (treeView1) مصدرا للعقد وليكن الثاني منهما مستقبلا للعقد treeView2
3. اذهب الى الخاصية AllowDrop من المستقبل (treeView2) واجعلها تاخذ القيمة true . في الواقع ان هذه الخاصية تسمح للمستقبل بتلقي المعلومات المسقطة عليه
4. اذهب الى الحدث ItemDrag من المكون المرسل treeView1 وقم بكتابة الكود التالي :

```
private void treeView1_ItemDrag(object sender, ItemDragEventArgs e)
        {
            DoDragDrop(e.Item, DragDropEffects.Move);
        }
```

ان هذا الحدث يتفعل فورا عند أي محاولة سحب (Drag) في المكون المرسل ومن ثم فإننا استدعاءنا للمنهج DoDragDrop نكون قد قمنا بتجهيز بيانات العقدة المراد نقلها وهذا يتمثل في البارامتر e.Item كذلك نكون قد حددنا العملية المصاحبة للسحب بواسطة البارامتر DragDropEffects.Move

نذهب الى الحدث DragEnter من المكون المستقبل treeView2 وقم بكتابة الكود

```
private void treeView2_DragEnter(object sender, DragEventArgs e)
        {
            e.Effect = DragDropEffects.Move;
        }
```

لاحظ ان المستقبل اصبح يعرف نوعية العملية المصاحبة للسحب (Dragging) والتي قام المرسل بها (DragDropEffects.Move)

5. الان نذهب الى الحدث DragDrop من المكون المستقبل ونكتب الكود التالي :

```
private void treeView2_DragDrop(object sender, DragEventArgs e)
 {
   if (e.Data.GetDataPresent("System.Windows.Forms.TreeNode", false))
            {
                Point pt = new Point(e.X,e.Y);
                pt=  treeView2.PointToClient(pt);
                TreeNode n = treeView2.GetNodeAt(pt);
                TreeNode n1 =
(TreeNode)e.Data.GetData("System.Windows.Forms.TreeNode");
                n.Nodes.Add((TreeNode)n1.Clone());
                n.ExpandAll();
                n1.Remove();
             }

   }
```

تقوم الشفرة السابقة بالاتي:

- اولا المنهج `e.Data.GetDataPresent`

البامتر الاول هو نوعية البيانات المراد التي يفترض ان المرسل اعدها

يختبر المنهج نوعية البيانات المعدة من قبل المرسل فاذا كان نوعها هو نفس`System.Windows.Forms.TreeNode`

فان المنهج يعيد true ويعيد False في الحالة الاخرى ..البارامتر الثاني يجعل المنهج يفحص التطابق فحسب اذا كانت قيمته False, اما اذا قيمته true فان المنهج يحصل على امكانية تحوير البيانات المعدة من قبل المرسل الى النوع الموجود في البارامتر الاولوهو امر استبعدناه في مثالنا هذا , أي اننا طلبنا منه فحص التطابق ولا نريد تحوير البيانات المعدة من قبل المرسل اطلاقا.

- بعد ذلك يقوم المستقبل بتحديد العقدة الواقعة تحت مؤشر الماوس ولكي يفعل ذلك لابد من تحديد الاحداثيات وفق المكون المستقبل وليس وفق الشاشة لان بارامتر الحدث **e** يعطي الاحداثيات وفق للشاشة بينما تحديد العقدة يحتاج الى احداثيات وفق الحدث....!؟

(الاحداثيات وفق الشاشة يعني اننا نحتسب البداية من الزاوية العليا اليسرى للشاشة بينما احتساب الاحداثيات وفق مكون معين معناها ان نحتسب البداية من الزاوية العليا اليسرى للمكون)

```
pt=  treeView2.PointToClient(pt);
```

ان هذا السطر يقوم بتحويل احداثيات مؤشر الفارة من نظام يبدا حسب الشاشة الى نظام يبدا حسب المكون`treeView2`

ويضع النتيجة داخل المتغير `pt`

```
TreeNode n = treeView2.GetNodeAt(pt);
```

ويقوم السطر السابق باخذ العقدة الواقعة تحت مؤشر الفارة من المكون المستقبل وستكون هذه العقدة **والدا** للعقدة القادمة من قبل المرسل.

```
TreeNode n1 =  (TreeNode)e.Data.GetData("System.Windows.Forms.TreeNode");
```

طبعا بعد تاكدنا من النوع المطلوب يطابق`TreeNode` فاننا سناخذ البيانات التي جاءنا بها المرسل ونضعها داخل متغير من النوع `TreeNode` وهو الامر الذي سيقوم به المنهج `GetData`.

انظر الى السطر الذي يلي السابق:

```
n.Nodes.Add((TreeNode)n1.Clone());
```

لاحظ ان هذا السطر يقوم بنسخ العقدة التي استقبلنا مع جميع ابناءها (`n1.Clone()`) ويجعلها كابن للعقدة التي كانت تحت مؤشرالفارة من في المستقبل (`n.Nodes.Add`).

قم بتشغيل البرنامج وتمتع بنقل العقد من الشجرة الاولى الى الشجرة الاخيرة