

SEPTEMBER 17, 2017

C# İLE MAPINFO OLE UYGULAMASI

.NET C# İLE X64 PLATFORMUNDA HARITA UYGULAMASI GELİŞTİRME
DÖKÜMANI

OMER UYGUN
BAŞARSOFT

İçerik

.Net C# İle MapInfo x64 OLE Uygulaması Geliştirme	2
Visual Studio İle Windows Form Uygulaması Oluşturulması.....	2
Gerekli Referansların Eklenmesi.....	2
Geliştirilen Uygulamaya MapInfo Entegrasyonunun Yapılması	5
Workspace in Uygulamaya Yüklenmesi.....	8
Dosya Menüsünün Geliştirilmesi.....	11
WorkSpace Aç.....	12
Tablo Aç.....	12
Tüm Tabloları Kapat	12
Tablo Kapat.....	12
Harita Araçları Menüsü Geliştirmesi	13
Select	13
Pan.....	13
Zoom In.....	13
Zoom Out.....	14
Bilgi	14
Çizim Araçları Menüsünün Geliştirilmesi.....	23
Nokta Çizim Aracı	23
Çizgi Çizim Aracı.....	26
Kapalı Alan Çizim Aracı	28
Güncelleme ve Silme İşlevlerinin Geliştirilmesi.....	31
Nokta Katmanı Güncelleme ve Silme İşlevi	31
Çizgi Katmanı Güncelleme ve Silme İşlevi	33
Kapalı Alan Katmanı Güncelleme ve Silme İşlevi	34
Genel Notlar	36
Not 1	36
Not 2	36

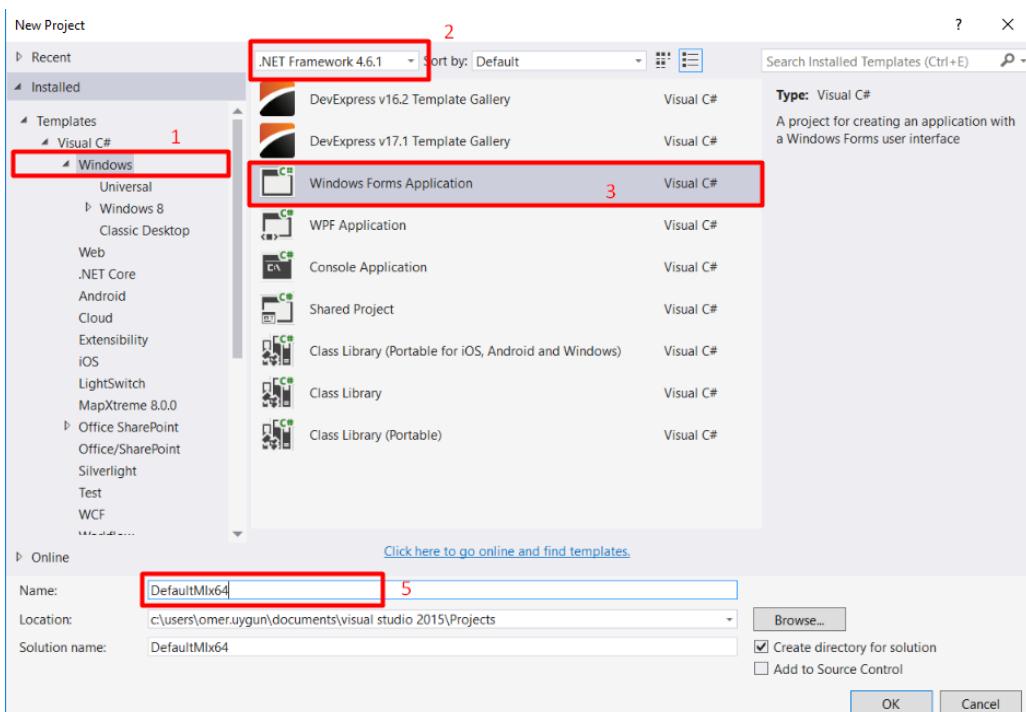
.Net C# ile MapInfo x64 OLE Uygulaması Geliştirme

C# ile MapInfo OLE uygulaması geliştirmek için geliştirme yapılan ortamda MapInfo nun kurulu olması gerekmektedir. MapInfo 16 uygulamasını kurulum talimatlarına göre kurduktan sonra uygulama geliştirmeye başlayabilirsiniz.

Visual Studio İle Windows Form Uygulaması Oluşturulması

Not: Dökümda kullanılan görseller Visual Studio 2015 Professionaldan alınmıştır. Bu nedenle Visual Studio 2013 ten veya diğer Visual Studio 2015 farklı versiyonları arasında farklılıklar gösterebilir.

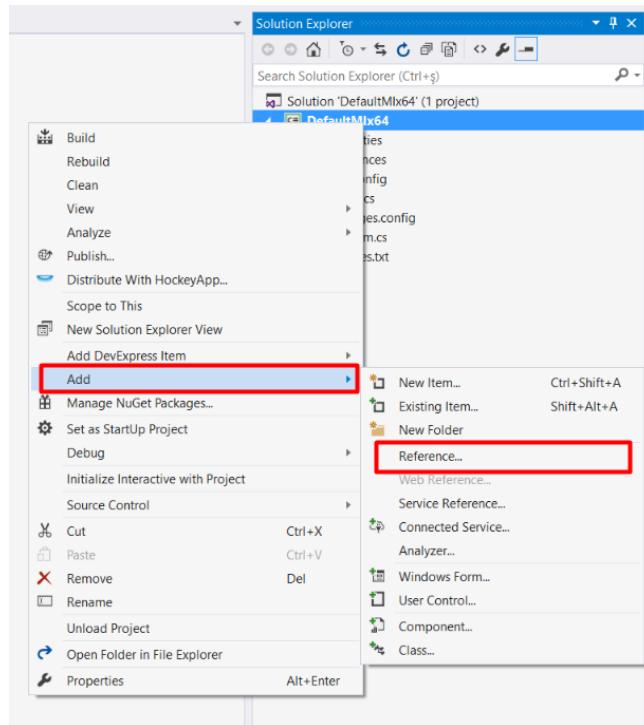
Visual Studio açılarak File> New> Project yolu yardımıyla yeni proje oluşturma ekranı açılır.



Bu ekrandan ilk olarak 1 adımı ile Windows seçili hale getirilir. İkinci adım olarak ise projenin geliştirilmesinde kullanılacak framework seçimi yapılır. Burada önemli olan nokta framework ün minumum olarak .Net Framework 4.6.1 olması gereğidir. Framework seçiminden sonra ise listelenen proje türleri içerisinde Windows Forms Application seçilir. En son olarak 5. Adımda olduğu gibi projeye bir isim verilerek OK butonu ile yeni proje oluşturulur.

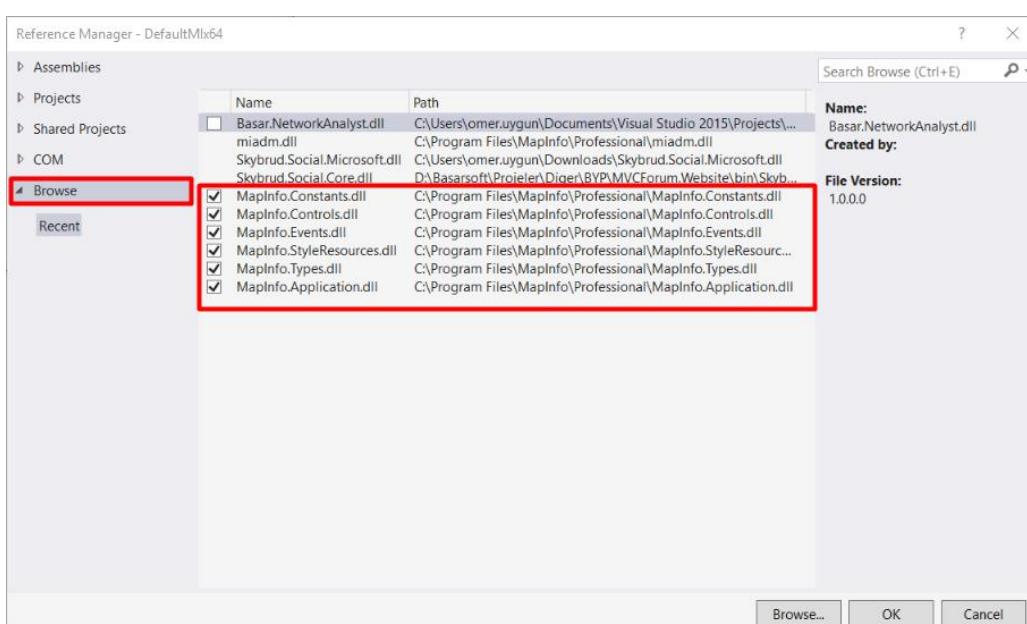
Gerekli Referansların Eklenmesi

OLE uygulası için gerekli referansların eklenmesi için, Visual Studio da proje üzerine sağ tık yapılarak Add > Reference seçeneği ile referans ekleme ekranı açılır.



İlk olarak MapInfo'nun referansları eklenmelidir. Bu referansları Browse diyerek "C:\Program Files\MapInfo\Professional" dizininden eklenebilir. Eklenmesi gereken MapInfo referansları aşağıdaki gibidir.

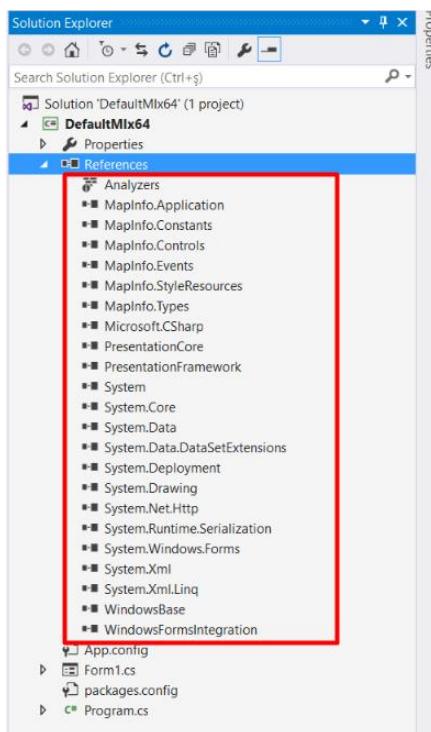
- ❖ MapInfo.Events.dll
- ❖ MapInfo.Constants.dll
- ❖ MapInfo.Controls.dll
- ❖ MapInfo.StyleResources.dll
- ❖ MapInfo.Types.dll
- ❖ MapInfo.Application.dll



MapInfo Referansları eklendikten sonra, MapInfo'nun kullandığı sistem referansları da yine aynı referans ekleme ekranda Assemblies sekmesinden eklenmelidir. Eklenmesi gereken referansların listesi aşağıdaki gibidir.

- ❖ PresentationCore
- ❖ PresentationFramework
- ❖ WindowsBase
- ❖ WindowsFromIntegration

Bu referanslarda eklendikten sonra Solution Explorer da Referanslarınız aşağıdaki resimdeki gibi olmalıdır.



Son olarak uygulamanın çalışabilmesi için bazı MapInfo dosyalarının geliştirdiğiniz uygulamanın exesinin olduğu dizine kopyalanması gerekmektedir. Kopyalanması gereken dosyaların listesi aşağıdaki gibidir. Bu dosyalar "C:\Program Files\MapInfo\Professional" dizininde yer almaktadır.

- ❖ acPDFCreatorLib.Net.dll
- ❖ Fdo.dll
- ❖ FDOCommon.dll
- ❖ FDOGeometry.dll
- ❖ MAPINFOPRO.MNU
- ❖ MAPINFOW.PRJ
- ❖ midfdo.dll
- ❖ midiOCI.dll
- ❖ midlodbc.dll
- ❖ Syncfusion.Shared.MVVM.Wpf.dll

- ❖ Xalan-C_11.dll
- ❖ XalanMessages_1_11.dll
- ❖ xerces-c_3_1.dll

Not: Geliştirdiğiniz uygulamanın exesine ulaşmak için Solution Explorer dan projenize sağ tıklayıp açılan menüden Open Folder in File Explorer seçeneğine tıkladıktan sonra açılan klasörden bin> Debug klasörüne hızlı bir şekilde ulaşabilirsiniz. Yukarıdaki dosyaları exenizin çalışacağı dizine kopyalandıktan sonra geliştirmeye başlayabilirsiniz.

Geliştirilen Uygulamaya MapInfo Entegrasyonunun Yapılması

İlk olarak oluşturduğunuz uygulama içerisinde yer alan Program.cs içerisinde MapInfo uygulamasının kurulu olduğu dizinin setlenmesi ve MapInfo uygulamasının başlatılması gerekmektedir. Bunun için Program.cs içerisinde yer alan “Main” metodu içerisinde aşağıdaki resimde olduğu gibi ilk olarak dll dizin setlemesi ardından da MapInfo başlatma işlemi yapılmalıdır. 1 numaralı alandaki kodlar ile MapInfo çalışma dizinini setlemek için kullanılmaktadır. Bu kodların hatasız çalışabilmesi için Program.cs içerisinde using satırına “`using MapInfo.Application;`” ve “`using System.Runtime.InteropServices;`” namespace leri eklenmelidir.

```
[DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
[return: MarshalAs(UnmanagedType.Bool)] 1

1 reference
static extern bool SetDllDirectory(string lpPathName);

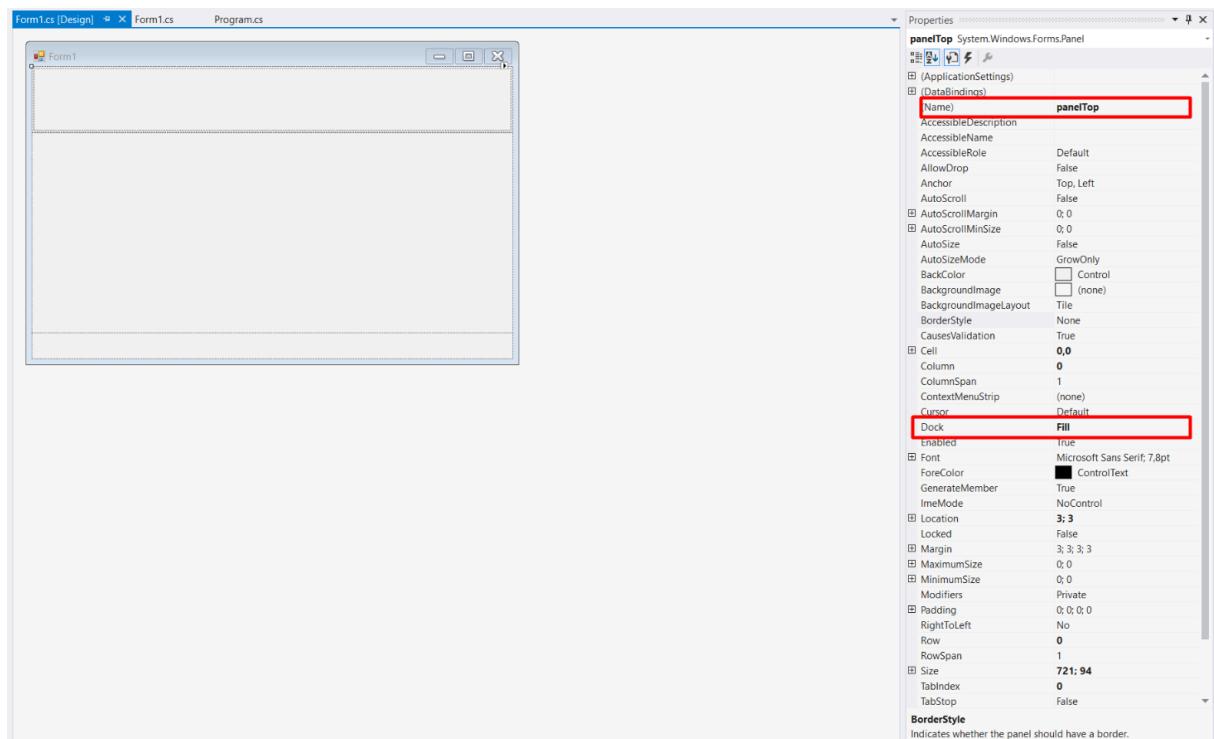
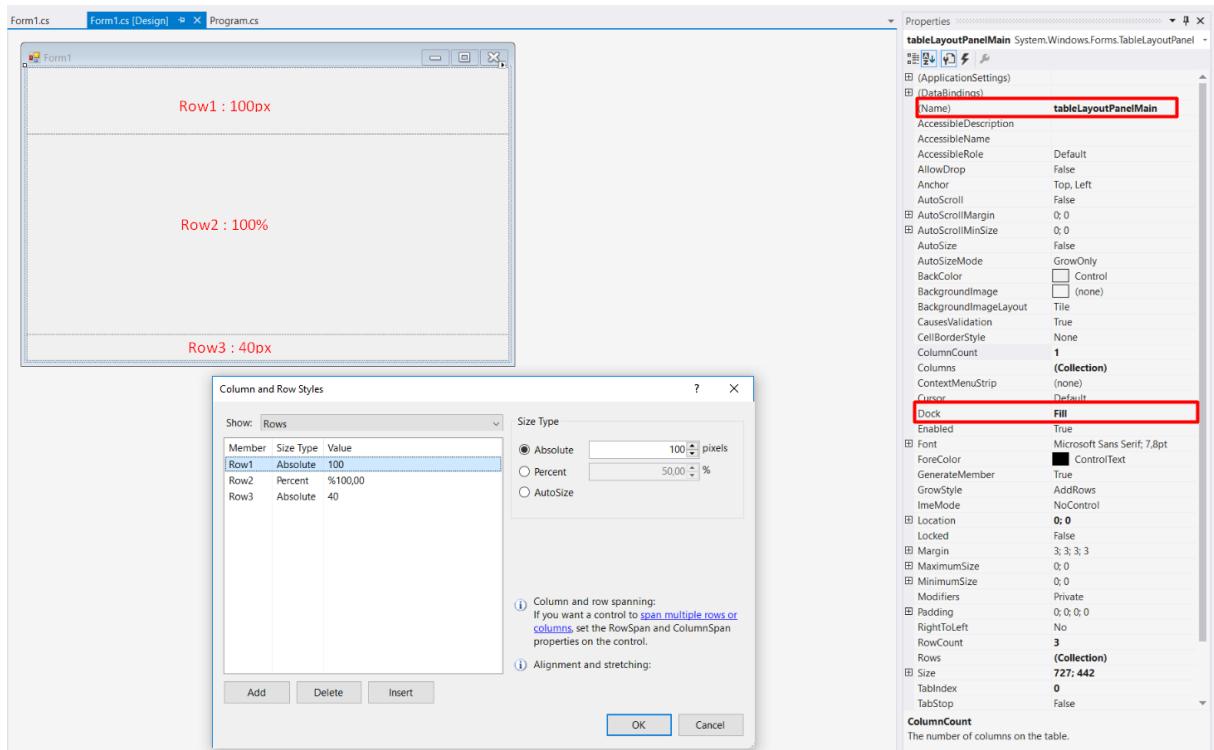
/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]
0 references
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

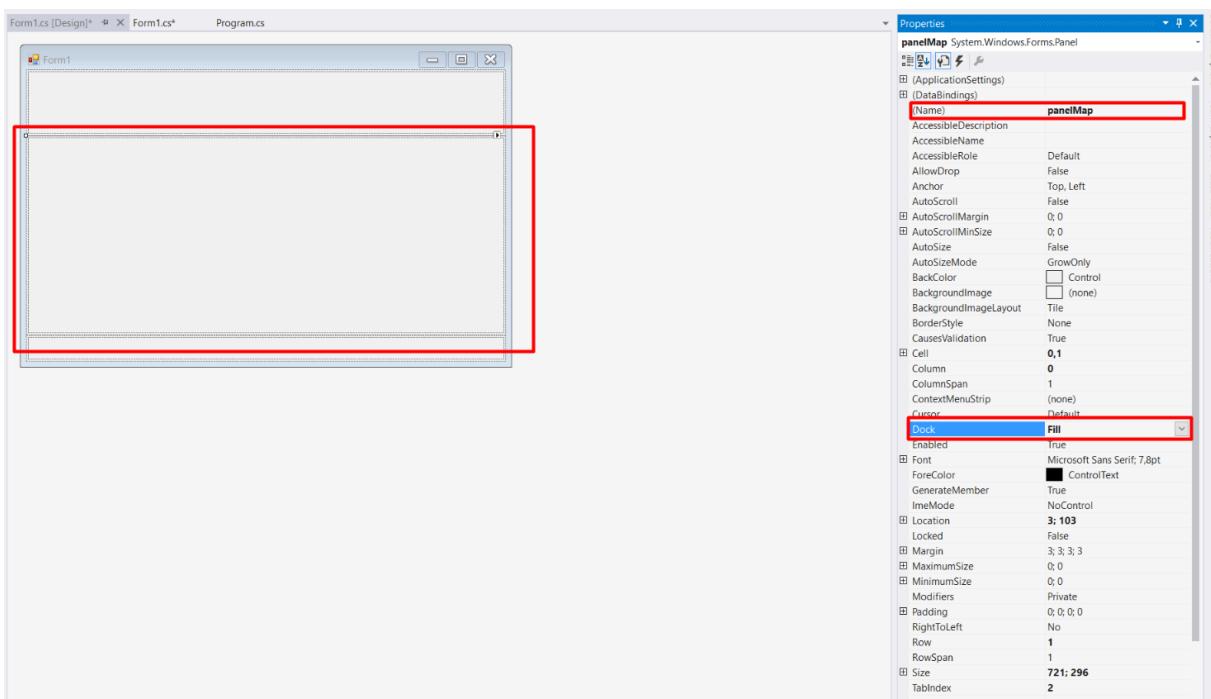
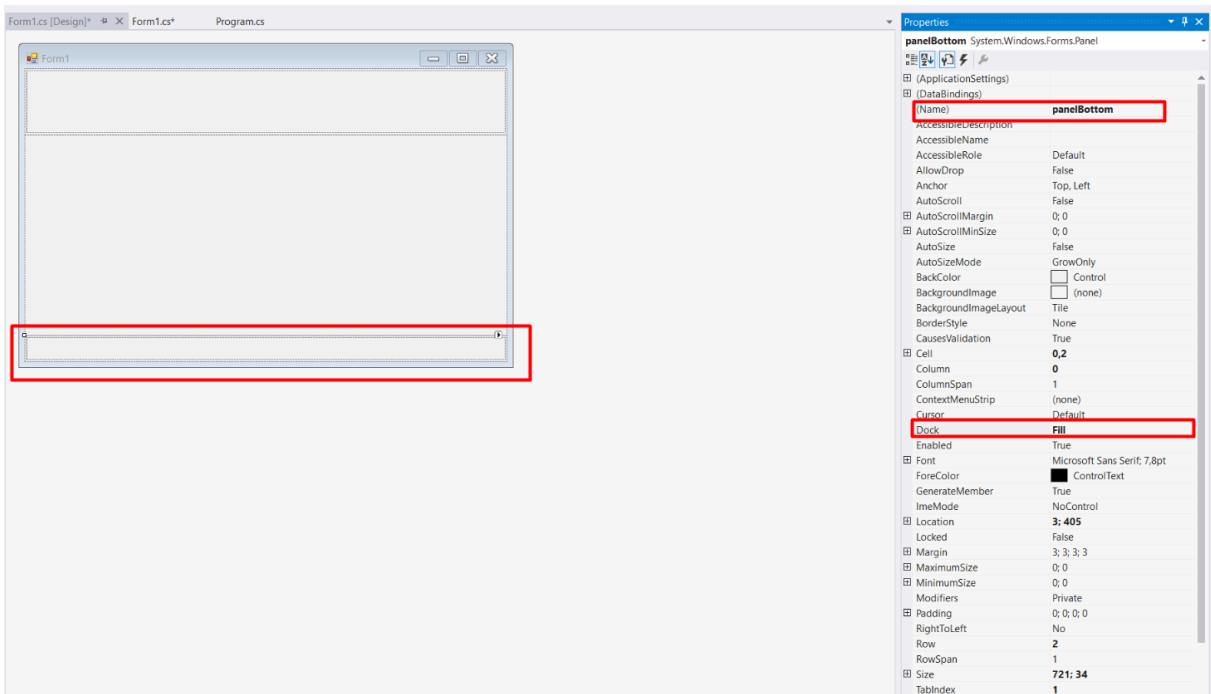
    //MapInfo kurulum dizinin setlenmesi 2
    SetDllDirectory(@"C:\Program Files\MapInfo\Professional");

    //MapInfo'nun başlatılması 3
    MapInfoCore.StartUp(null);
}

Application.Run(new Form1());
}
```

Oluşturulan proje de yer alan Form1 formumuzu design modda açıyoruz. Bunun için Solution Explorer içerisinde yer alan Form1.cs dosyasına çift tıklamanız yeterlidir. Form arayüzümüzde hem harita hemde işlemleri gerçekleştireceğimiz butonları koyabilmek için 3 adet panel eklememiz gerekmektedir. Panelleri form boyutu otomatik değiştiğinde yeniden boyutlanması için TableLayoutPanel içerisinde eklenmesinde fayda vardır. Bunun için ilk olarak TableLayoutPanel eklenmeli, TableLayoutPanel’ın 3 satır 1 sütun olacak şekilde konfigüre edilmesi gerekmektedir. TableLayoutPanel’ın örnek ekran görüntüsü aşağıdaki gibidir. Bu panelleri TableLayoutPanel içerisindeki satırlara sürükleyip bırakarak hepsinin Dock pozisyonunu Fill olarak setliyoruz.





Panelleri TableLayoutPanel içerisine ekledikten sonra arka Form1 için MapInfo entegrasyonunu yapabiliyoruz. Bunun için Form1.cs içerisinde ilk olarak harita görüntüsünü form içerisinde gösterebilme için formumuzu "[IIIntegratedMappingApplication](#)" interfaceinden kalıtım alacak şekilde düzenlememiz gerekmektedir. Bunun için aşağıdaki şekilde classımıza interface imizi ekliyoruz. Ardından interface in metodlarını implemente etmemiz gerekmektedir. Bunun için aşağıdaki resimde kırmızı ile işaretli olan "[IIIntegratedMappingApplication](#)" a tıklayıp kılavyeden Ctrl+. Tuş kombinasyonu ile açılan menüden imlement interface seçeneğine basmanız yeterli olacaktır. Bu şekilde visual studio otomatik olarak interface i form1.cs içerisinde implement edecektir.

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using MapInfo.Types;
using System.Windows.Forms.Integration;
using System.Windows;

namespace DefaultMIX64
{
    public partial class Form1 : Form, IIntegratedMappingApplication
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Interface i implement ettikten sonra otomatik oluşan metodlar içerisinde “WindowCreated” metoduna harita görüntümüzü uygulamamızda gösterebilme için gerekli olan kontrolleri eklememiz gerekiyor. Bunun için aşağıdaki resimde görüldüğü gibi panelMap isimli panelimize harita görüntü kontrolümüzü ekliyoruz.

```
0 references
public void WindowCreated(IWindowInfo window, IEnumerable<KeyValuePair<string, string>> properties)
{
    var wi = window as MapInfo.Types.WindowInfo;
    if (wi == null)
        return;

    var ctrl = new ElementHost
    {
        Child = wi.UserControl,
        Dock = DockStyle.Fill
    };
    if (wi.IsDocumentWindow)
    {
        panelMap.Tag = wi.WindowId;
        panelMap.Controls.Add(ctrl);
    }
}
```

Workspace in Uygulamaya Yüklenmesi

MapInfo taraflı işlemler yapabilmek için MapInfo kütüphanesinin bir instanceını global değişken olarak tanımlamamız gerekmektedir. Bunun için ilk olarak Program.cs içerisinde static olarak aşağıdaki şekilde MapInfo instance değişken olarak tanımlıyoruz.

```

namespace DefaultMIX64
{
    static class Program
    {
        public static MapInfo.Types.IMapInfoApplication miApplication { get; set; }

        [DllImport("kernel32.dll", CharSet = CharSet.Unicode, SetLastError = true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        static extern bool SetDllDirectory(string lpPathName);
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);

            //MapInfo kurulum dizinin setlenmesi
            SetDllDirectory(@"C:\Program Files\MapInfo\Professional");

            //MapInfonun başlatılması
            MapInfoCore.StartUp(null);
        }
    }
}

```

Daha sonrasında bu tanımladığımız değişkene i Form1.cs sınıfında Load metodu içerisinde aşağıdaki şekilde yeni bir instance alarak değer ataması yapıyoruz.

```

namespace DefaultMIX64
{
    public partial class Form1 : Form, IIIntegratedMappingApplication
    {
        public Form1()
        {
            InitializeComponent();
        }

        IIIntegratedMappingApplication Interface i metodları

        private void Form1_Load(object sender, EventArgs e)
        {
            Program.miApplication = MapInfoCore.Initialize(Handle, this);
        }
    }
}

```

Değişken tanımlamasından sonra workspace'ımızın dizin bilgisini bir değişkene tanımlıyoruz. Aşağıdaki resimde workspace imiz uygulamamızın çalıştığı dizinin içindeki MI_DATA klasörü içerisinde olduğu için uygulama çalışma dizini + workspace olacak şekilde tanımlamasını yapıyoruz. Ardından Program.cs içerisinde tanımladığımız miApplication değişkeninde yer alan “RunMapBasicCommand” metodu ile workspace yüklememizi yapıyoruz.

```

namespace DefaultMlx64
{
    public partial class Form1 : Form, IIntegratedMappingApplication
    {
        public Form1()
        {
            InitializeComponent();
        }

        IIntegratedMappingApplication Interface i metodları

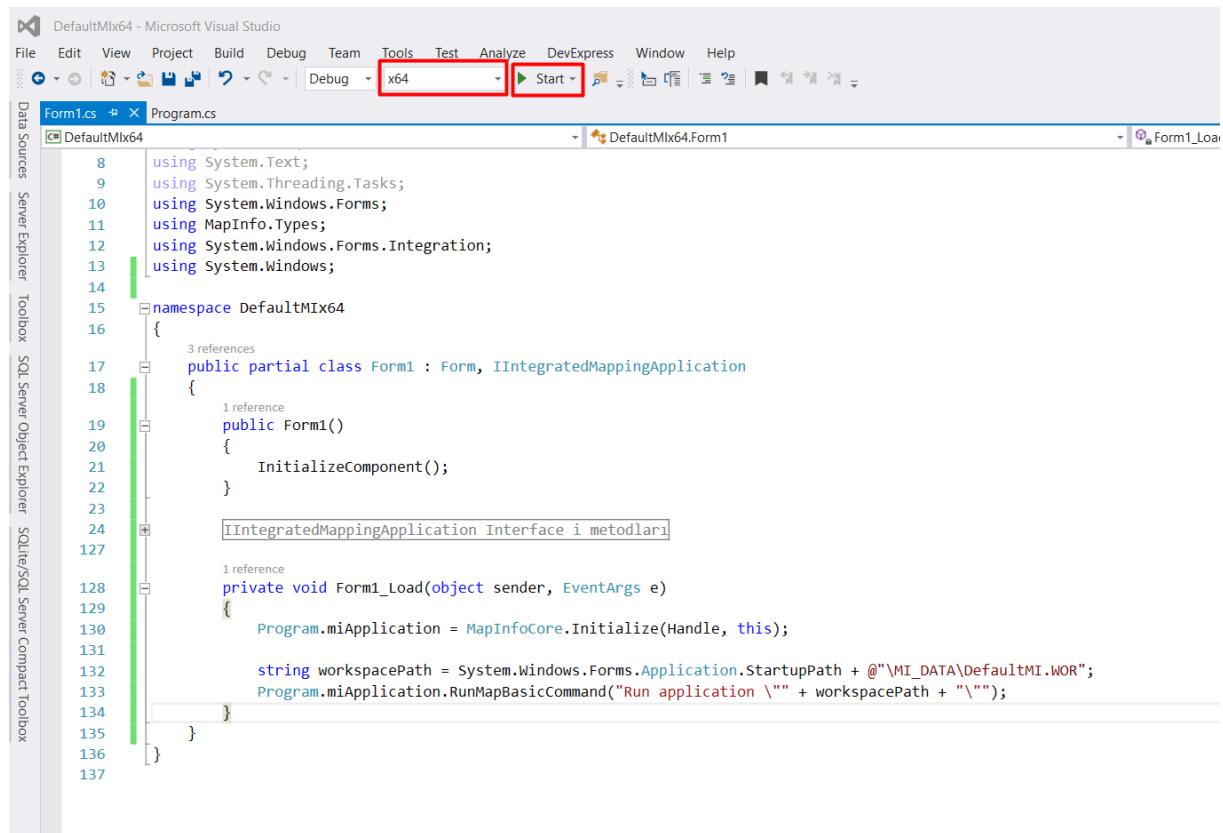
        private void Form1_Load(object sender, EventArgs e)
        {
            Program.miApplication = MapInfoCore.Initialize(Handle, this);

            string workspacePath = System.Windows.Forms.Application.StartupPath + @"\MI_DATA\DefaultMI.WOR";
            Program.miApplication.RunMapBasicCommand("Run application \" " + workspacePath + "\"");
        }
    }
}

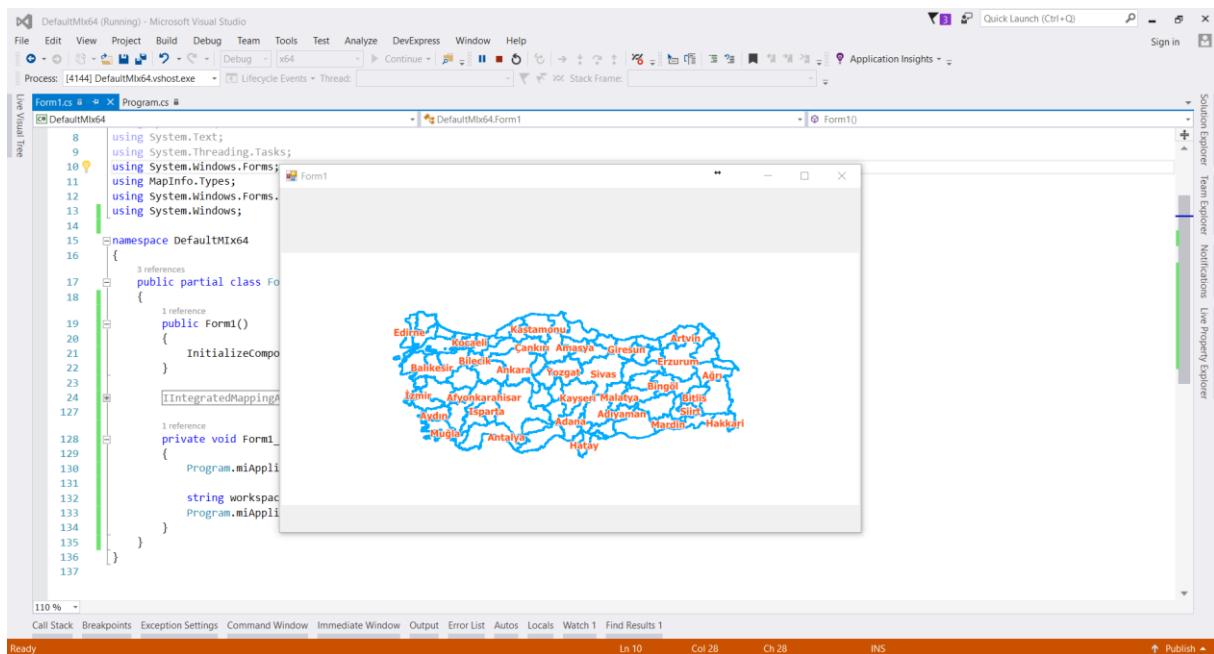
```

Not: MapInfo x64 platformu üzerinde koştuğu için uygulamamız x64 platformunda build edilmelidir.

Aşağıdaki resimde görüldüğü üzere “Start” butonu ile uygulamamızı çalıştırarak workspace yüklememizi test edebiliriz.

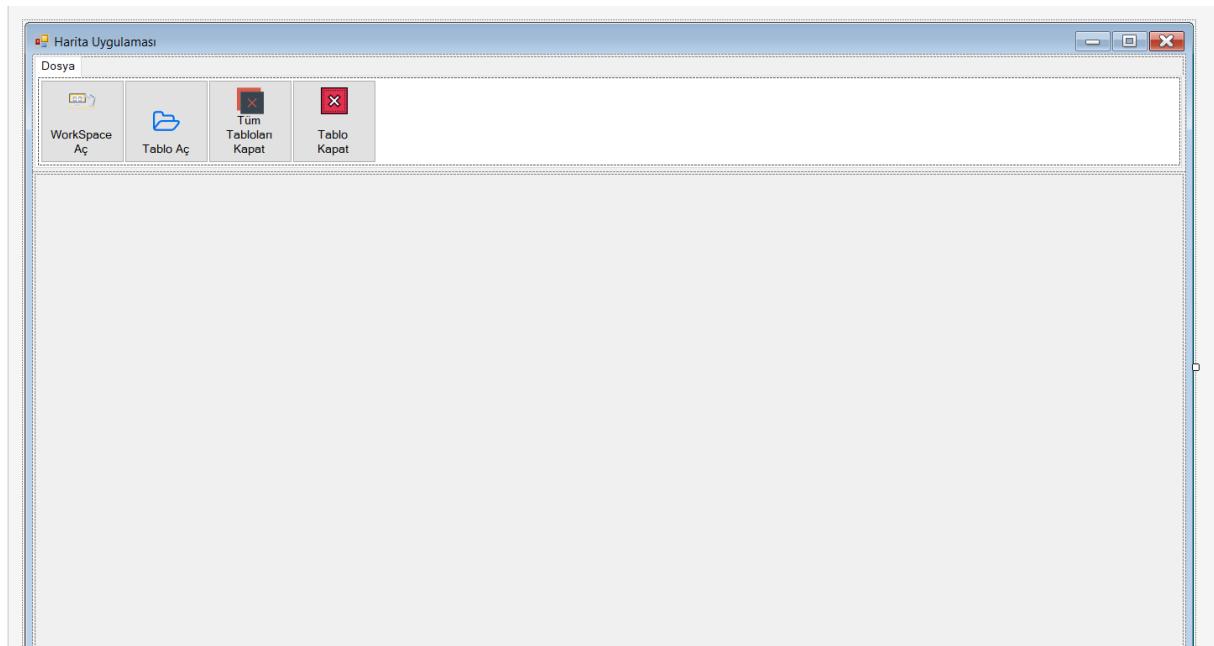


Uygulamamız başarılı bir şekilde derlendikten sonra aşağıdaki ekran görüntüsündeki gibi harita görüntüsü form arayüzüne gelecektir.



Dosya Menüsünün Geliştirilmesi

Dosya menümüzde yer alacak “Workspace Aç”, “Tablo Aç”, “Tüm Tabloları Kapat”, “Tablo Kapat” işlevleri için Form1 üzerinde aşağıdaki şekilde bir tasarım oluşturuyoruz.



Tasarımızın ardından MapInfo nun genel araçlarını kullanabilmek için Form1.cs içerisinde aşağıdaki metodu oluşturuyoruz.

```
0 references
public void ExecuteCommand(MapInfoCommand cmd, object sender)
{
    if (cmd.CanExecute(sender, null))
        cmd.Execute(sender, null);
}
```

Not: “`MapInfoCommand`” nesnesini kullanabilmek için using satırına “`using MapInfo.Commands;`” namespaceinin eklenmesi gerekmektedir.

WorkSpace Açı

Uygulamamızın yüklenmesi anında yüklediğimiz workspace haricinde yeni bir workspace i açmak istersek aşağıdaki şekilde “Workspace Açı” butonumuzun click event’ina aşağıdaki kodu yazmamız yeterli olacaktır. Bu metotda yer alan “`ApplicationCommands`” MapInfo kütüphanesinin standart arayüzlerini açmak için kullanılmaktadır. Workspace açmak için bu statik sınıfın “`OpenWorkspace`” property’si kullanılmaktadır.

```
1 reference
private void buttonWorkSpaceAc_Click(object sender, EventArgs e)
{
    ExecuteCommand(ApplicationCommands.OpenWorkspace, sender);
}
```

Tablo Açı

Uygulamamızın çalışma anında yeni bir tablo açmak istediğimizde aşağıdaki kod ile MapInfo Tablo Açı aracı üzerinden yeni bir tablo açabiliriz. Bunun için workspace açmadaki yöntem ile bu sefer “`OpenTable`” property’si ile tablo açma komutumuzu çalıştıracak kodumuzu aşağıdaki şekilde yazıyoruz.

```
1 reference
private void buttonTabloAc_Click(object sender, EventArgs e)
{
    ExecuteCommand(ApplicationCommands.OpenTable, sender);
}
```

Tüm Tabloları Kapat

Uygulamamızda tabloları kapatmak için kullanılacak butonun kodunu workspace açma işlemindeki gibi aşağıdaki şekilde ekliyoruz.

```
1 reference
private void buttonTumTablolariKapat_Click(object sender, EventArgs e)
{
    ExecuteCommand(ApplicationCommands.CloseAll, sender);
}
```

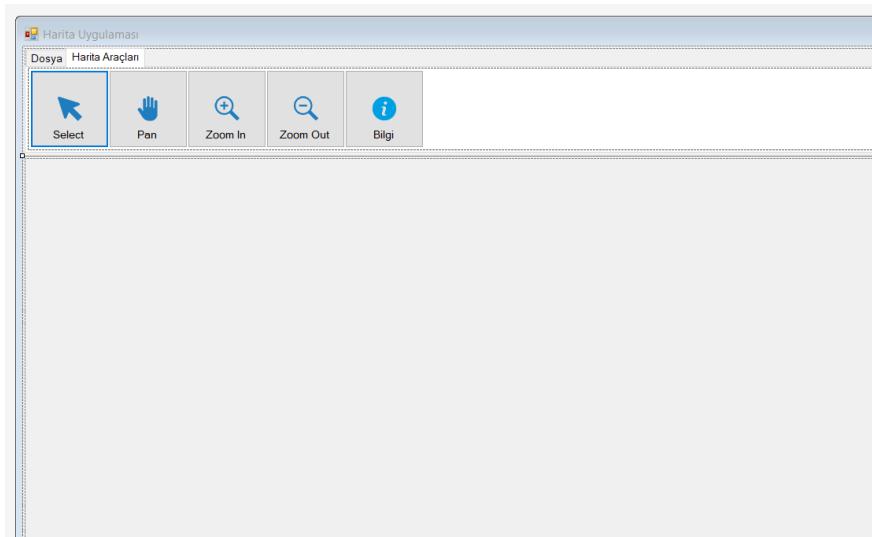
Tablo Kapat

Uygulamamızda seçimli olarak tablo kapatmak için kullanılacak butonun kodunu yine workspace açma işlemindeki gibi aşağıdaki şekilde ekliyoruz.

```
1 reference
private void buttonTabloKapat_Click(object sender, EventArgs e)
{
    ExecuteCommand(ApplicationCommands.CloseTable, sender);
}
```

Harita Araçları Menüsü Geliştirmesi

Bu menüdeki araçlar harita üzerindeki temel işlemler olan “Select”, “Zoom In”, “Zoom Out”, “Pan” ve Bilgi almak için kullanılacak “Bilgi” araçları geliştirilecektir. Bunun için aşağıdaki ekran görüntüsünde olduğu gibi “Harita Araçları” menümüzün tasarımını yapıyoruz.



Select

Harita üzerinde obje veya objeleri seçili hale getirmek için bu araç kullanılmaktadır. Bu araç MapInfo nun standart araçlarındanandır. Bu nedenle aracı aktifleştirmek için “[ToolCommands](#)” sınıfındaki “Select” property’si çalıştırılmalıdır.

```
1 reference
private void buttonSelect_Click(object sender, EventArgs e)
{
    ExecuteCommand(ToolCommands.Select, sender);
}
```

Pan

Harita üzerinde kaydırma işlemi için bu araç kullanılmaktadır. Bu araç MapInfo nun standart araçlarındanandır. Bu nedenle aracı aktifleştirmek için “[ToolCommands](#)” sınıfındaki “Recenter” property’si çalıştırılmalıdır.

```
1 reference
private void buttonPan_Click(object sender, EventArgs e)
{
    ExecuteCommand(ToolCommands.Recenter, sender);
}
```

Zoom In

Harita üzerinde yaklaşma işlemi için bu araç kullanılmaktadır. Bu araç MapInfo nun standart araçlarındanandır. Bu nedenle aracı aktifleştirmek için “[ToolCommands](#)” sınıfındaki “ZoomIn” property’si çalıştırılmalıdır.

```
1 reference
private void buttonZoomIn_Click(object sender, EventArgs e)
{
    ExecuteCommand(ToolCommands.ZoomIn, sender);
}
```

Zoom Out

Harita üzerinde uzaklaşma işlemi için bu araç kullanılmaktadır. Bu araç MapInfo nun standart araçlarındanandır. Bu nedenle aracı aktifleştirmek için “[ToolCommands](#)” sınıfındaki “ZoomOut” property’si çalıştırılmalıdır.

```
1 reference
private void buttonZoomOut_Click(object sender, EventArgs e)
{
    ExecuteCommand(ToolCommands.ZoomOut, sender);
}
```

Bilgi

Bilgi aracı olarak MapInfo standart aracı yerine özelleştirilmiş bir araç kullanılacaktır. Bu aracın çalışma mantığı diğer özelleştirilmiş araçların kullanım mantığı ile benzerlik gösterecektir. Bu nedenle Bilgi aracı bazı ekstra geliştirmelerin yapılması gerekmektedir. Kodlarımızın daha kolay okunabilirliğinin sağlanması ve özelleştirilmiş işlemlerimizin bir sınıf içerisinde toplanması için projemize “[CustomToolButtons](#)” adında bir sınıf ekliyoruz. İlk olarak bu sınıfımız ile birlikte kullanacağımız “[DrawingModeDef](#)”, “[ButtonName](#)”, “[IconDef](#)” enumları ile “[DelegateCommand](#)”, “[CustomListItem](#)” sınıflarını projemiz içerisinde oluşturup içeriklerini aşağıdaki şekilde dolduruyoruz.

```
namespace DefaultMix64
{
    0 references
    public enum DrawingModeDef
    {
        /// <summary>
        /// Nokta(Point) çizimi Aracı İçin Kullanılmaktadır. Bu araç ile harita üzerine nokta atarak çizim yapılabilir.
        /// </summary>
        DM_CUSTOM_POINT = 34,
        /// <summary>
        /// Çizgi(Line) çizimi yapmak için kullanılmaktadır. Bu araç ile harita üzerine 2 noktadan oluşan çizgi nesnesi çizilebilir.
        /// </summary>
        DM_CUSTOM_LTINE = 33,
        /// <summary>
        /// Kare/Dikdörtgen(Rect) çizimi yapmak için kullanılmaktadır. Bu araç ile harita üzerine simetrik olarak Kare/Dikdörtgen çizimi yapılabilir.
        /// </summary>
        DM_CUSTOM_RECT = 32,
        /// <summary>
        /// Daire(Circle) çizimi yapmak için kullanılmaktadır. Bu araç ile harita üzerine simetrik olarak daire çizimi yapılabilir.
        /// </summary>
        DM_CUSTOM_CIRCLE = 30,
        /// <summary>
        /// Elips(Ellipse) çizimi yapmak için kullanılmaktadır. Bu araç ile simetrik bir şekilde elips çizimi yapılabilir.
        /// </summary>
        DM_CUSTOM_ELLIPSE = 31,
        /// <summary>
        /// Kapalı Alan(Region) çizimi yapmak için kullanılmaktadır. Bu araç ile istenilen şekilde kapalı alan oluşturulabilir.
        /// </summary>
        DM_CUSTOM_POLYGON = 35,
        /// <summary>
        /// Çoklu çizgi(Polyline) çizimi yapmak için kullanılmaktadır. Bu araç ile 2 den fazla noktadan oluşan çoklu çizgi nesnesi oluşturulabilir.
        /// </summary>
        DM_CUSTOM_POLYLINE = 36,
    }
}

namespace DefaultMix64
{
    /// <summary>
    /// Özelleştirilen çizim araçlarının tanımlanıldığı sabit dosyasıdır. Uygulama girişinde kullanılacak özelleştirilmiş araçlar MapInfo ya yüklenmektedir.
    /// </summary>
    0 references
    public enum ButtonName
    {
        /// <summary>
        /// Çizgi çizim aracı
        /// </summary>
        AddLine,
        /// <summary>
        /// Bilgi aracı
        /// </summary>
        GetInfo,
        /// <summary>
        /// Kapalı alan çizim aracı
        /// </summary>
        AddPolygon,
        /// <summary>
        /// Nokta çizim aracı
        /// </summary>
        AddPoint
    }
}
```

```

]namespace DefaultMIX64
{
    /// <summary>
    /// Cizim araçları aktifleştirildiğinde cursorun nasıl görüneceğini belirlemek için kullanılmaktadır.
    /// </summary>
    0 references
}public enum IconDef
{
    MI_CURSOR_ARROW = 0,
    MI_CURSOR_IBeam_CROSS = 135,
    MI_CURSOR_HELP = 137,
}
]

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

]namespace DefaultMIX64
{
    4 references
}public class DelegateCommand : ICommand
{
    private readonly Predicate<object> _canExecute;
    private readonly Action<object> _execute;

    public event EventHandler CanExecuteChanged;

    0 references
}public DelegateCommand()
{
}

1 reference
}public DelegateCommand(Action<object> execute)
    : this(execute, null)
{
}

1 reference
}public DelegateCommand(Action<object> execute,
    Predicate<object> canExecute)
{
    _execute = execute;
    _canExecute = canExecute;
}

```

```

0 references
public bool CanExecute(object parameter)
{
    if (_canExecute == null)
    {
        return true;
    }

    return _canExecute(parameter);
}

0 references
public void Execute(object parameter)
{
    _execute(parameter);
}

0 references
public void RaiseCanExecuteChanged()
{
    if (CanExecuteChanged != null)
    {
        CanExecuteChanged(this, EventArgs.Empty);
    }
}
}

using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DefaultMIx64
{
    11 references
    public class CustomListItem
    {
        8 references
        public string Text { get; set; }
        8 references
        public object Value { get; set; }
        5 references
        public CustomListItem()
        {
            this.Text = string.Empty;
        }
        0 references
        public CustomListItem(string text, object value)
        {
            this.Text = text;
            this.Value = value;
        }
        7 references
        public override string ToString()
        {
            return Text;
        }
    }
}

```

CustomToolButtons sınıfımızda işlemleri hızlandırmak ve tekrar tekrar instance oluşturmadan çalışabilme için aşağıdaki şekilde gerekli olan değişkenlerimizi oluşturuyoruz.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DefaultMIX64
{
    3 references
    public class CustomToolButtons
    {
        #region Tanımlamalar
        /// <summary> Harita görüntümüze bastığımız formun bir örneği. Bu örnek çizim araçlarının aktifleştirilmesinde kullanılm ...
        0 references
        private System.Windows.Forms.Form _mainForm { get; set; }

        /// <summary> MapInfo taraflında oluşturduğumuz özel araçların listesini C# taraflında tutmak için kullanılmaktadır.
        public Dictionary<string, int> _dic = new Dictionary<string, int>();

        /// <summary> Hızlı bir şekilde işlemler yapabilmek için sınıfın kendisinden statik değişken oluşturarak kullanılmaktadır ...
        5 references
        private static CustomToolButtons instance { get; set; }
        #endregion

        0 references
        public static CustomToolButtons Instance
        {
            get
            {
                if (instance != null)
                    return instance;
                else
                {
                    instance = new CustomToolButtons();
                    return instance;
                }
            }
            set
            {
                instance = value;
            }
        }
    }
}

```

CustomToolButtons sınıfımız içerisinde ilk olarak oluşturduğumuz özelleşmiş çizim araçlarını yüklemek için kullanacağımız metodumuzu oluşturuyoruz. Bu metodumuz harita görüntüsünü bastığımız formun instanceını alacak şekilde bir metod olmalıdır. Bu metodumuza da “LoadCustomTools” ismini veriyoruz ve gönderilen formu CustomToolButtons içerisinde oluşturduğumuz değişkene atamasını yapıyoruz.

```

        .
        .
        .
        /// <summary> Özelleştirilmiş araçları yüklemek için kullanılmaktadır.
        0 references
        public void LoadCustomTools(System.Windows.Forms.Form mainForm)
        {
            this._mainForm = mainForm;
        }

```

Daha sonrasında oluşturacağımız bütün özelleştirilmiş çizim araçlarını eklemek için kullanacağımız metodumuzu oluşturuyoruz. Bu metodumuza da “LoadCustomToolBar” ismini verdikten sonra parametreleri ve içeriğini aşağıdaki şekilde olacak şekilde gerekli düzenlemeyi yapıyoruz.

```

/// <summary>
/// Özelleştirilmiş aracın MapInfo ya tanıtılmazı için kullanılmaktadır.
/// </summary>
/// <param name="buttonName">Cizim aracımızın isim bilgisi</param>
/// <param name="methodName">Cizim işlemi tamamlandıktan sonra MapInfo'nun c# tarafında çalıştıracağı metodun adı</param>
/// <param name="commandParam">Parametreler</param>
/// <param name="drawingModeDef">Cizim aracımızın tür bilgisini içermektedir</param>
/// <param name="mapCursorId">Cizim aracı aktifleştirildikten sonra fare imlecinin alacağı stil bilgisini içerir</param>
/// <param name="bModifierKey"></param>
1 reference
public void LoadCustomToolBar(ButtonName buttonName, Action<object> methodName, object commandParam, DrawingModeDef drawingModeDef, int mapCursorId, bool bModifierKey = false)
{
    _dic.Add(buttonName.ToString(), new DelegateCommand(methodName,
    null,
    (int)drawingModeDef,
    mapCursorId.ToString(),
    bModifierKey));
}

```

Bu metodumuzda tamamladıktan sonra uygulama açılışında özelleştirilmiş araçlarımızı etkinleştirmek için Form1.cs sınıfındaki Load metodu içerisinde aşağıdaki şekilde “LoadCustomTools” metodumuzu çağırıyoruz. Bu şekilde Form1.cs içerisinde çok fazla kod karmaşıklığına girmeden özelleştirilmiş araçlarımızı yükleyebiliyor olacağız.

```

1 reference
private void Form1_Load(object sender, EventArgs e)
{
    Program.miApplication = MapInfoCore.Initialize(Handle, this);

    string workspacePath = System.Windows.Forms.Application.StartupPath + @"\MI_DATA\DefaultMI.WOR";
    Program.miApplication.RunMapBasicCommand("Run application \"\" + workspacePath + "\"");

    CustomToolButtons.Instance.LoadCustomTools(this);
}

```

Evet şimdi özelleştirilmiş bilgi al aracının geliştirmesi adımına geçebiliriz. Bunun için ilk olarak “CustomToolButtons” sınıfı içerisinde bilgi almak için harita üzerine tıkladıktan sonra çalışacak olan metodumuzu ekliyoruz. Bu metodun ismini de “GetInfo” olarak belirliyoruz.

```

/// <summary>
/// Bilgi al işlevi için. Harita üzerine tıklama yapıldıktan sonra MapInfo tarafından bu metod çalıştırılmaktadır.
/// </summary>
/// <param name="param"></param>
1 reference
public void GetInfo(object param)
{
}

```

Sonraki adımda “CustomToolButtons” sınıfı içerisinde oluşturduğumuz “LoadCustomTools” metodu içerisinde “Bilgi Al” aracımızın tanımlamasını yapıyoruz. Aşağıdaki resimde kırmızı ile işaretlenen alan içerisindeki kod ile Bilgi aracımızı MapInfo ya tanıtıyoruz.

```

/// <summary> Özelleştirilmiş araçları yüklemek için kullanılmaktadır.
1 reference
public void LoadCustomTools(System.Windows.Forms.Form mainForm)
{
    this.mainForm = mainForm;
    LoadCustomToolBar(ButtonName.GetInfo, GetInfo, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_HELP);
}

```

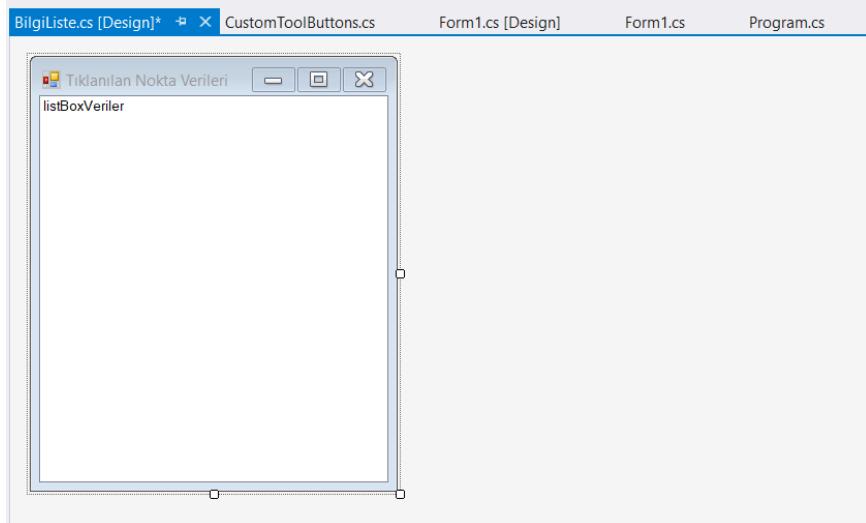
Not: Bilgi aracında harita üzerine tıkladıktan sonra “GetInfo” metodumuzun çalışacağını aracımızı MapInfo ya tanıtırken aşağıdaki parametre ile belirtmişik.

```

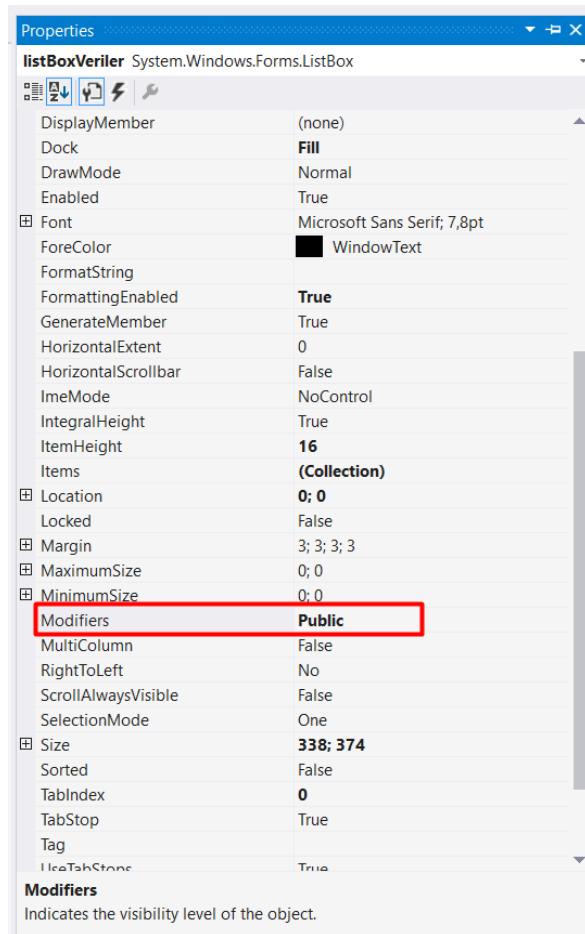
/// <summary> Özelleştirilmiş araçları yüklemek için kullanılmaktadır.
1 reference
public void LoadCustomTools(System.Windows.Forms.Form mainForm)
{
    this._mainForm = mainForm;
    LoadCustomToolBar(ButtonText.GetInfo, GetInfo, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_HELP);
}
...

```

Şimdi tıklama işlemi sonrasında tıklanılan noktadaki verileri bulmak için gerekli olan kodları yazmaya başlayabiliriz. Bilgi alma işlemindeki mantık, “eğer tıklanılan noktada tek bir veri var ise ilgili verinin güncelleme formu açılır, eğer tıklanılan noktada birden fazla veri var ise ilk olarak kullanıcıya bu veriler listelenir ardından kullanıcı liste üzerinden bir kayıt seçerek seçilen ilgili olduğu güncelleme ekranı açılır” şeklindedir. Bu algoritmada ilk olarak birden fazla kayıt olması durumunda kayıtların listelenmesi için projemize yeni bir tane “BilgiListe” adında form ekliyoruz. Daha sonrasında formumuza bir adet listbox controlü ekliyoruz. Eklediğimiz listbox kontrolümüzün Dock property’sini Fill olarak ayarladıkta sonra aşağıdaki ekran görüntüsünde olduğu gibi tasarım elde etmiş oluyoruz.



Listbox kontrolümüzün diğer formlardan da erişilebilmesi için Modifiers property sini “Public” olarak değiştirmemiz gerekiyor.



Liste üzerinden seçilen kaydın detay bilgisinin gösterilebilmesi için listBox kontrolümüzün doubleClick metodu içerisinde aşağıdaki kodları yazıyoruz. Aşağıdaki resimde kırmızı ile işaretlenen global değişlenimize seçili kaydın bilgisi atanmaktadır. Bu bilgi daha sonra “CustomToolButtons” sınıfı içerisindeki “GetInfo” metodunda kullanılacaktır, bu nedenle global olarak tanımlanmıştır.

```

4 references
public partial class Bilgiliste : Form
{
    2 references
    public string SeciliKayit { get; set; }

    1 reference
    public Bilgiliste()
    {
        InitializeComponent();
    }

    1 reference
    private void listBoxVeriler_MouseDoubleClick(object sender, MouseEventArgs e)
    {
        try
        {
            if (listBoxVeriler.SelectedIndex >= 0)
            {
                CustomListItem secili = (CustomListItem)listBoxVeriler.Items[listBoxVeriler.SelectedIndex];
                if (secili != null)
                {
                    this.SeciliKayit = secili.Value.ToString();
                    this.DialogResult = DialogResult.OK;
                    this.Close();
                }
            }
        }
        catch (Exception ex)
        {
        }
    }
}

```

Evet şimdilik bilgi al işlevi için gerekli olan kodlarımızı “`CustomToolButtons`” sınıfı içerisindeki “`GetInfo`” metoduna eklemeye geldi. `GetInfo` Metodunda ilk olarak harita üzerinde tıklanılan noktanın “x” ve “y” koordinatlarının alınması gerekmektedir. X ve Y koordinatlarının alınabilmesi için `MapInfo` metodu olan “`commandinfo`” metodu kullanılmaktadır. Bu metoda 1 parametresini göndererek x koordinatı, 2 parametresi gönderilerek y koordinatı alınabilir. Ardından bu noktadaki veriler “`SearchPoint`” `MapInfo` metodu ile sorgulanabilir. Bu metod 3 parametre almaktadır. İlk parametre harita gösterimini yaptığımız pencerenin `MapInfo` tarafından id bilgisi, ikincisi x koordinatı, üçüncü ise y koordinatıdır. Aşağıdaki örnekte kullanılan “`frontwindow()`” `MapInfo` metodu harita görüntüsünün id bilgisini döndürmektedir.

Not: `MapInfo` kütüphanesi üzerinde işlem yaparken en çok kullanılan 2 metod “`EvalMapBasicCommand`” ve “`RunMapBasicCommand`” dir. “`EvalMapBasicCommand`” metodu geri bir değer döndürmek istendiğinde, “`RunMapBasicCommand`” metodu ise sorgu çalıştırmak için kullanılmaktadır.

Eğer tıklanılan noktada kayıt var ise bu kayıtlar üzerinde kayıt sayısı kadar dönerken kaydın hangi tabloda ve kaydın tablodaki rowid bilgisi ile listede göstermek için bir sütununun değeri alınarak listeye eklenmektedir. Daha sonrasında döngü sonrasında listedeki elemanların sayısına bakılarak eğer kayıt sayısı 0 veya daha küçük ise kullanıcıya mesaj verilmekte, 1 ise direk kaydın güncelleme ekranı, eğer 1 den fazla ise kullanıcının detayı görmek istediği kaydı seçmesi için “BilgiListe” formu açılmaktadır.

```
/// Bilgi al işlevi için. Harita Üzerine tıklama yapıldıktan sonra MapInfo tarafından bu metod çalıştırılmaktadır.
/// </summary>
/// <param name="param"></param>
1reference
public void GetInfo(object param)
{
    try
    {
        List<CustomListItem> tıklanılanNoktadakiVeriler = new List<CustomListItem>();

        string x = Program.miApplication.EvalMapBasicCommand("commandinfo(1)").Replace(',', '.'); // Harita Üzerinde tıklanılan noktanın x değeri alınıyor.
        string y = Program.miApplication.EvalMapBasicCommand("commandinfo(2)").Replace(',', '.'); // Harita Üzerinde tıklanılan noktanın y değeri alınıyor

        // Tıklanılan noktadaki kayıtların sayısı alınıyor
        int tıklanılanNoktadakiKayitSayisi = Convert.ToInt32(Program.miApplication.EvalMapBasicCommand("SearchPoint(frontwindow()," + x + "," + y + ")"));

        if (tıklanılanNoktadakiKayitSayisi > 0)
        {
            for (int i = 1; i <= tıklanılanNoktadakiKayitSayisi; i++)
            {
                // Tıklanılan noktadaki verinin tablo adı bilgisi alınıyor.
                string tabloAdi = Program.miApplication.EvalMapBasicCommand("SearchInfo(" + i.ToString() + ",1)");

                // Tıklanılan noktadaki verinin tablodaki rowid bilgisi alınıyor. Bu rowid bilgisi MapInfo tablosunda görülebilen bir alan değildir.
                // MapInfo'nun arka planda tuttuğu uniq bir sütun değeridir.
                string rowId = Program.miApplication.EvalMapBasicCommand("SearchInfo(" + i.ToString() + ",2)");

                // Tıklanılan Noktadaki kaydın bilgilerine erişebilmek için kaydı seçiyoruz
                Program.miApplication.RunMapBasicCommand("Fetch rec " + rowId + " From " + tabloAdi);

                // Eğer bir işlem sonunda herhangi bir şekilde açık kalmış bir selection tablomuz var ise bunlar üzerinde işlem yapmamak için eklenmiştir.
                if (!tabloAdi.StartsWith("Sel_"))
                {
                    switch (tabloAdi)
                    {
                        case "CIZGI_KATMANI":
                            tıklanılanNoktadakiVeriler.Add(new CustomListItem()

```

```

        {
            Text = tabloAdi + " (" + Program.miApplication.EvalMapBasicCommand(tabloAdi + ".ADI") + ")",
            Value = tabloAdi + "#" + rowId
        });
        break;
    case "IL":
        tikanilanNoktadakiVeriler.Add(new CustomListItem()
        {
            Text = tabloAdi + " (" + Program.miApplication.EvalMapBasicCommand(tabloAdi + ".IL_ADI") + ")",
            Value = tabloAdi + "#" + rowId
        });
        break;
    case "ILCE":
        tikanilanNoktadakiVeriler.Add(new CustomListItem()
        {
            Text = tabloAdi + " (" + Program.miApplication.EvalMapBasicCommand(tabloAdi + ".ILCE_ADI") + ")",
            Value = tabloAdi + "#" + rowId
        });
        break;
    case "KAPALI_ALAN_KATMANI":
        tikanilanNoktadakiVeriler.Add(new CustomListItem()
        {
            Text = tabloAdi + " (" + Program.miApplication.EvalMapBasicCommand(tabloAdi + ".ADI") + ")",
            Value = tabloAdi + "#" + rowId
        });
        break;
    case "NOKTA_KATMANI":
        tikanilanNoktadakiVeriler.Add(new CustomListItem()
        {
            Text = tabloAdi + " (" + Program.miApplication.EvalMapBasicCommand(tabloAdi + ".ADI") + ")",
            Value = tabloAdi + "#" + rowId
        });
        break;
    default:
        break;
    }
}
}

if (tikanilanNoktadakiVeriler.Count <= 0)
{
    #region Eğer Tıklanılan Noktada Hiç Veri Bulunamaz ise
    System.Windows.Forms.MessageBox.Show("Tıkladığınız noktada hiç veri bulunamamıştır.");
    #endregion
}
else if (tikanilanNoktadakiVeriler.Count == 1)
{
    #region Eğer Tıklanılan Noktada Tek Bir Veri Var ise İlgili Kaydın Güncelleme Ekranı Açıılır
    Program._form.ShowInfo(tikanilanNoktadakiVeriler[0].Value.ToString());
    #endregion
}
else
{
    #region Eğer Tıklanılan Noktada Birden Fazla Veri Var İse Kullanıcının Bu Verilerden İstediğini Seçebileceği Listeleme Ekranı Açılmaktadır
    BilgiListe listeForm = new BilgiListe();
    listeForm.Owner = Program._form;
    listeForm.listBoxVeriler.Items.Clear();
    for (int i = 0; i < tikanilanNoktadakiVeriler.Count; i++)
    {
        listeForm.listBoxVeriler.Items.Add(tikanilanNoktadakiVeriler[i]);
    }
    DialogResult res = listeForm.ShowDialog();
    if (res == DialogResult.OK)
    {
        Program._form.ShowInfo(listeForm.SeciliKayit);
    }
    #endregion
}
}
catch (Exception ex)
{
    System.Windows.Forms.MessageBox.Show("Beklenmeyen bir hata oluştu.");
}
}

```

Detay Bilgisinin gösterilmesi için her yerden erişim sağlanabilmesi için harita formumuzda(Form1) “ShowInfo” metodunu oluşturuyoruz. Bu metodumuzda da switch-case yapısı ile katman bazında kontrolümüzü yaparak ilgili kaydın güncelleme formu açılmaktadır. Bu metodumuzun içeriği diğer çizim araçlarından sonra anlatılacaktır fakat metodun örnek yapısı aşağıdaki gibi olmalıdır.

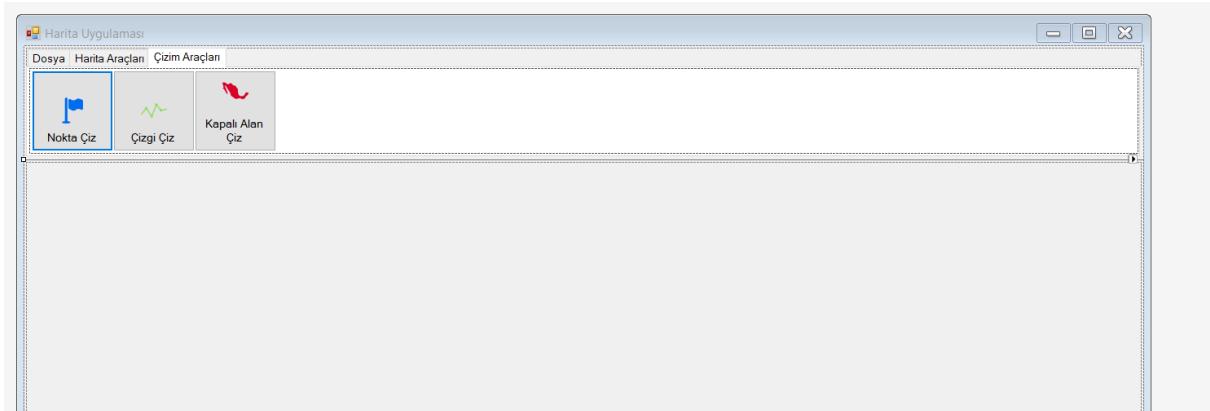
```

    /// <summary>
    /// Tıklanılan Noktadaki Verilerin Detay Bilgilerini Gösteren Metod
    /// </summary>
    /// <param name="key">KATMAN_ADI:ROW_ID</param>
    2 references
    public void ShowInfo(string key)
    {
        if (key.Split('#').Length == 2)
        {
            string tabloAdi = key.Split('#')[0];
            string rowId = key.Split('#')[1];
            switch (key)
            {
                case "IL":
                    #region İl Bilgisi Alma
                    #endregion
                    break;
                case "ILCE":
                    #region İlçe Bilgisi Alma
                    #endregion
                    break;
                case "NOKTA_KATMANI":
                    #region Nokta Katmanı Bilgi Alma
                    #endregion
                    break;
                case "CIZGI_KATMANI":
                    #region Çizgi Katmanı Bilgi Alma
                    #endregion
                    break;
                case "KAPALI_ALAN_KATMANI":
                    #region Kapalı Alan Katmanı Bilgi Alma
                    #endregion
                    break;
            }
        }
    }
}

```

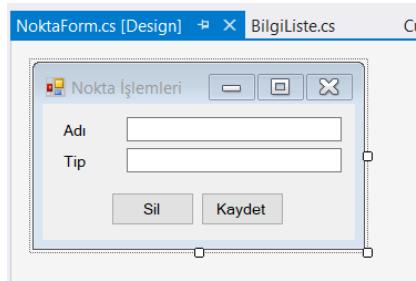
Çizim Araçları Menüsünün Geliştirilmesi

Bu araçlar özelleştirilmiş çizim araçlarını içermektedir. Bu menüden Nokta, Çizgi ve Kapalı Alan Çizimi araçları geliştirmesi yapılacaktır. Bu nedenle Çizim Araçları menüsünün tasarımları aşağıdaki şekilde yapıyoruz.



Nokta Çizim Aracı

Bu aracın geliştirmesinden önce ekleme ve güncelleme işlemlerinde kullanacağımız nokta formumuzu oluşturmamız gerekiyor. Bunun için Solution Explorer içerisinde “NoktaForm” adında bir form oluşturuyoruz. Daha sonrasında form arayüzümüzü aşağıdaki şekilde oluşturuyoruz. “NOKTA_KATMANI” tablomuzda “Tip” ve “Adı” adında iki tane alanımız olduğu için formumuza veri girişi için iki tane textbox ve 2 tane buton ekliyoruz. Örnek tasarımımız aşağıda gibidir.



Formumuzu oluşturduktan sonra çizim aracımızın tanımlamasına geçebiliriz. Bunun için “CustomToolButtons” sınıfı içerisinde çizim işlemi tamamlandıktan sonra çalışacak metodumuzu ekliyoruz. Bu metodumuz içerisinde bir tane obje tanımlıyoruz. Objemizi tıklanılan noktadan yeni bir obje olarak oluşturup bu objeye stil ataması yapıyoruz. Daha sonrasında bu objenizi tablomaşa yeni bir kayıt olarak ekliyoruz. Ardından diğer bilgilerin girilip kaydedilmesi için kaydımızı tablodan seçip geçici bir tabloya aktarıyoruz. Daha sonrasında kullanıcının bilgileri girmesi için “NoktaForm” adında oluşturduğumuz formumuzu çağırıyoruz.

```
/// <summary> Nokta çizim işleminde, Harita Üzerine nokta eklendikten sonra MapInfo tarafından çalıştırılacak metoddur. 
public void NoktaEkle(object param)
{
    // Yeni bir obje tanımlıyoruz.
    Program.miApplication.RunMapBasicCommand("dim o as object");
    // Tanımlanan objeye tıklanılan noktanın koordinat bilgileri ile yeni bir nokta oluşturup atamasını yapıyoruz.
    // Burada CommandInfo(1) ile tıklanılan noktadaki x koordinatını vermektedir.
    // Burada CommandInfo(2) ile tıklanılan noktadaki y koordinatı vermektedir.
    Program.miApplication.RunMapBasicCommand("o=createPoint(commandinfo(1),commandinfo(2))");
    //Objenin stil bilgisi değiştirilmektedir.
    Program.miApplication.RunMapBasicCommand("alter object o info 2, makesymbol(34, 16776960, 12)");
    // Oluşturulan obje katmana insert ediliyor.
    Program.miApplication.RunMapBasicCommand("insert into NOKTA_KATMANI(obj) values(o)");
    // Eklenen son kayıt tablodan seçiliyor.
    Program.miApplication.RunMapBasicCommand("fetch last from NOKTA_KATMANI");
    // Seçilen kayıtın rowid bilgisi alınıyor
    var rowId = Program.miApplication.EvalMapBasicCommand("NOKTA_KATMANI.rowid");
    // Alınan rowid bilgisi ile tablodan kayıt seçilerek geçici tablolaya aktarılıyor
    Program.miApplication.RunMapBasicCommand("Select * from NOKTA_KATMANI where rowid=" + rowId + " into Sel_Nokta neselect");
    // Tip ve Adı bilgilerinin girilmesi için NoktaForm u oluşturuluyor.
    NoktaForm form = new NoktaForm();
    // Bu form aynı şekilde güncelleme işleminde de kullanılacağı için form içerisinde Global bir değişken oluşturularak bu değişkenin değeri false olarak ayarlanıyor.
    form.IsUpdate = false;
    // Nokta formumuzu açıyoruz.
    DialogResult res = form.ShowDialog();

    // Eğer kullanıcı kaydet butonu dışında formu direk kapatırsa geçici olarak açılan tablo kapatılıyor.
    if (res != DialogResult.OK)
    {
        Program.miApplication.RunMapBasicCommand("rollback table Sel_Nokta");
        Program.miApplication.RunMapBasicCommand("close table Sel_Nokta");
    }
    // Yukarıda oluşturulan obje kaldırılıyor.
    Program.miApplication.RunMapBasicCommand("undim o");
}
```

“NoktaForm” adındaki formumuz içindeki “Kaydet” butonumuzun Click metoduna aşağıdaki kodlarımızı yazıyoruz. Burada boş geçilip geçilmeme kontrolü yapılarak kaydetme işlemi tamamlanmaktadır.

```

1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace DefaultMlx64
12 {
13     public partial class NoktaForm : Form
14     {
15         public bool IsUpdate { get; set; }
16         public NoktaForm()
17         {
18             InitializeComponent();
19         }
20
21         private void NoktaForm_Load(object sender, EventArgs e)...
22
23         private void buttonKaydet_Click(object sender, EventArgs e)
24         {
25             if (textBoxAdi.Text.Trim().Length <= 0)
26             {
27                 MessageBox.Show("Lütfen Adı bilgisini giriniz.");
28                 return;
29             }
30             if (textBoxTip.Text.Trim().Length <= 0)
31             {
32                 MessageBox.Show("Lütfen Tip bilgisini giriniz.");
33             }
34             // Adı ve Tip bilgileri güncellenmekte
35             Program.miApplication.RunMapBasicCommand("Update Sel_Nokta set ADI=\"" + textBoxAdi.Text + "\", TIP=\"" + textBoxTip.Text + "\"");
36             // Güncellemeyi kaydediyoruz
37             Program.miApplication.RunMapBasicCommand("commit table Sel_Nokta");
38             // Açıtığımız geçici tabloya kapatıyoruz
39             Program.miApplication.RunMapBasicCommand("close table Sel_Nokta");
40             MessageBox.Show("Kayıt işlemi başarılı bir şekilde yapılmıştır.");
41             this.DialogResult = DialogResult.OK;
42             this.Close();
43         }
44     }
45 }

```

Kaydetme işlemi için kodlarımızı tamamladıktan sonra aracımızı MapInfo'ya tanıtabilir ve Form1 de bu aracı aktifleştirmek için kodlarımızı yazabiliriz. Öncelikli olarak aracımızı MapInfo'ya tanıtıyoruz. Bunun için “[CustomToolButtons](#)” sınıfı içerisindeki “LoadCustomTools” metodumuzda bilgi alma aracında tanımladığımız gibi nokta ekle aracımızı tanımlıyoruz.

```

1 reference
public void LoadCustomTools(System.Windows.Forms.Form mainForm)
{
    this._mainForm = mainForm;
    LoadCustomToolBar(ButtonText.GetInfo, GetInfo, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_HELP);
    LoadCustomToolBar(ButtonText.AddPoint, NoktaEkle, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_ARROW);
}

```

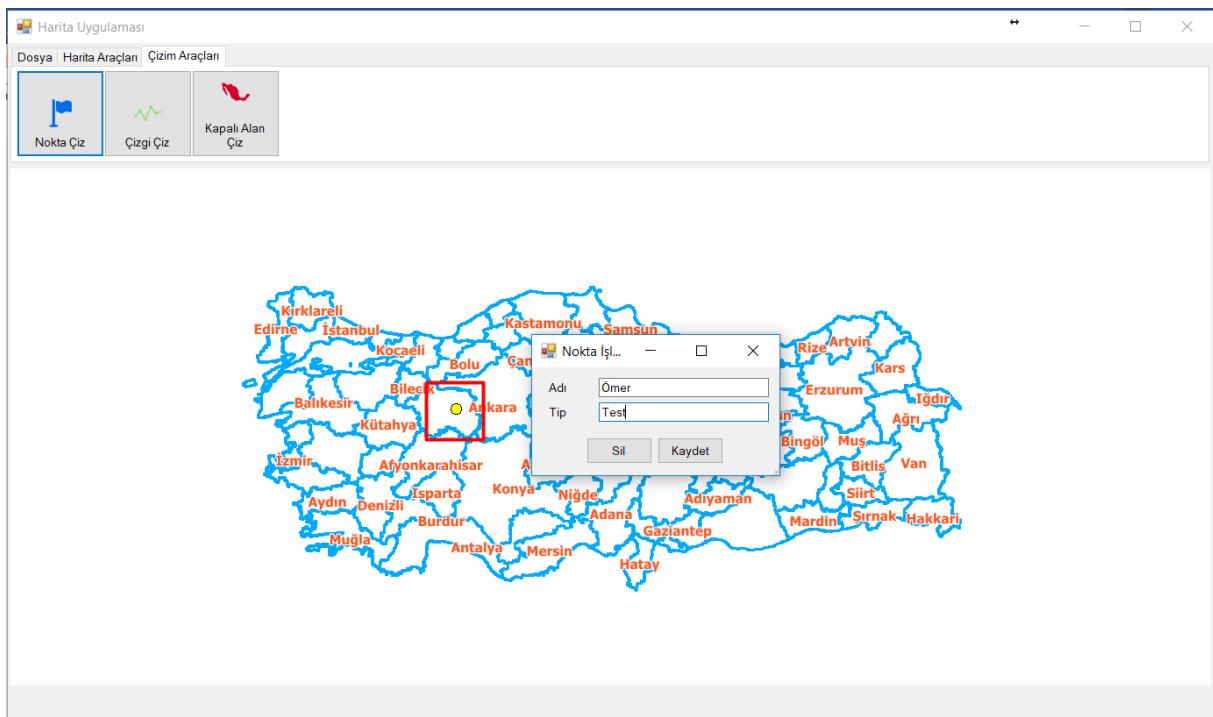
Şimdi sıra geldi aracımızı aktifleştirecek kodumuzu yazmaya. Bunun için Nokta Çizim için oluşturduğumuz butonumuzun Click metoduna aşağıdaki kod ekliyoruz.

```

1 reference
private void buttonNoktaCizim_Click(object sender, EventArgs e)
{
    CustomToolButtons.Instance.ActivateCustomTool(ButtonText.AddPoint);
}

```

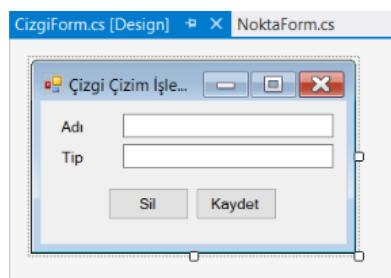
Kodumuzu çalıştırıldıkten sonra örnek nokta ekleme işlemi sonucumuz aşağıdaki gibi olacaktır.



Çizgi Çizim Aracı

Çizgi çizim aracının geliştirmesi de nokta çizim aracının geliştirmesinin aynısı olarak yapılmaktadır. Burada tekbir farklı nokta bulunmaktadır. Bu nokta Nokta çizimde kullanılan “`commandinfo(1)`” ve “`commandinfo(2)`” metodlarına ek olarak “`commandinfo(3)`” ve “`commandinfo(4)`” metodları ile objenin bitiş x ve y koordinatı alınmakta ve bu noktalardan “`CreateLine`” metodu ile yeni bir line objesi oluşturularak “`o`” objesine atama yapılmaktadır. Geri kalan bütün geliştirme adımları aynıdır. Aşağıda sırasıyla kodlar yer almaktadır.

Çizgi çizim işlemleri için aşağıdaki şekilde bir form tasarımi oluşturulmuştur.



Ardından çizim işlemi tamamlandıktan sonra MapInfo tarafından çalıştırılacak metodumuz “`CustomToolButtons`” sınıfı içerisinde aşağıdaki gibi oluşturulmuştur.

```

/// <summary> Çizgi çizim işlevinde. Harita üzerine çizgi çizimi yapıldıktan sonra MapInfo tarafından çalıştırılacak met ...
Reference
public void CizgiEkle(object param)
{
    // Yeni bir obje tanımlıyoruz.
    Program.miApplication.RunMapBasicCommand("dim o as object");
    // Tanımlanan objeyi yapılan çizimin x ve y değerleri alınarak yeni bir line objesi oluşturularak ataması yapılmaktadır.
    Program.miApplication.RunMapBasicCommand("o=CreateLine(commandinfo(1),commandinfo(2),commandinfo(5),commandinfo(6))");
    //Objemin stil bilgisi değiştirilmektedir.
    Program.miApplication.RunMapBasicCommand("alter object o info 2, makepen (2,2,16711680)");
    // Oluşturulan obje katmana insert ediliyor.
    Program.miApplication.RunMapBasicCommand("insert into CIZGI_KATMANI(obj) Values(o)");
    // Eklenen son kayıt tablodan seçiliyor.
    Program.miApplication.RunMapBasicCommand("fetch last from CIZGI_KATMANI");
    // Seçilen kaydan rowid bilgisi alınıyor
    var rowId = Program.miApplication.EvalMapBasicCommand("CIZGI_KATMANI.rowid");
    // Alınan rowid bilgisi ile tablodan kayıt seçilerek geçici tabloya aktarılıyor
    Program.miApplication.RunMapBasicCommand("Select * from CIZGI_KATMANI where rowid=" + rowId + " into Sel_Cizgi neselect");
    // Tip ve Adı bilgilerinin girilmesi için Cizgi formu oluşturuluyor.
    CizgiForm form = new CizgiForm();
    // Bu form aynı şekilde güncelleme işleminde de kullanılacağı için form içerisinde Global bir değişken oluşturularak bu değişkenin değeri false olarak ayarlanıyor.
    form.IsUpdate = false;
    // Nokta formumuzu açıyoruz.
    DialogResult res = form.ShowDialog();
    // Eğer kullanıcı kaydet butonu dışında formu direk kapatırsa geçici olarak açılan tablo kapatılıyor.
    if (res != DialogResult.OK)
    {
        Program.miApplication.RunMapBasicCommand("rollback table Sel_Cizgi");
        Program.miApplication.RunMapBasicCommand("close table Sel_Cizgi");
    }
    // Yukarıda oluşturulan obje kaldırılıyor.
    Program.miApplication.RunMapBasicCommand("undim o");
}

```

Ardından çizgi formunda yer alan Kaydet butonun Click metodu içerisinde aşağıdaki resimde görüldüğü gibi oluşturulmuştur.

```

CizgiForm.cs  CizgiForm.cs [Design]  NoktaForm.cs  NoktaForm.cs [Design]  BilgiListe.cs  CustomToolButtons.cs  BilgiListe.cs [Design]  Form1.cs [Design]
DefaultMix64
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace DefaultMix64
12 {
13     public partial class CizgiForm : Form
14     {
15         public bool IsUpdate { get; set; }
16         public CizgiForm()
17         {
18             InitializeComponent();
19         }
20
21         private void CizgiForm_Load(object sender, EventArgs e)
22         {
23             if (textBoxAdi.Text.Trim().Length < 0)
24             {
25                 MessageBox.Show("Lütfen Adı bilgisini giriniz.");
26                 return;
27             }
28             if (textBoxTip.Text.Trim().Length < 0)
29             {
30                 MessageBox.Show("Lütfen Tip bilgisini giriniz.");
31             }
32             // Adı ve Tip bilgileri güncellenmekte
33             Program.miApplication.RunMapBasicCommand("Update Sel_Cizgi set ADI=\"" + textBoxAdi.Text + "\", TIP=\"" + textBoxTip.Text + "\"");
34             // Güncellenen verileri kaydediyoruz
35             Program.miApplication.RunMapBasicCommand("commit table Sel_Cizgi");
36             // Açığımız geçici tabloyu kapatıyor
37             Program.miApplication.RunMapBasicCommand("close table Sel_Cizgi");
38             MessageBox.Show("Kayıt işlemi başarılı bir şekilde yapılmıştır.");
39             this.DialogResult = DialogResult.OK;
40             this.Close();
41         }
42     }
43 }

```

Ardından çizgi çizim aracımız aşağıdaki gibi tanımlanmıştır.

```

/// <summary> Özelleştirilmiş araçları yüklemek için kullanılmaktadır.</summary>
1 reference
public void LoadCustomTools(System.Windows.Forms.Form mainForm)
{
    this._mainForm = mainForm;
    LoadCustomToolButton(ButtonName.GetInfo, GetInfo, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_HELP);
    LoadCustomToolButton(ButtonName.AddPoint, NoktaEkle, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_ARROW);
    LoadCustomToolButton(ButtonName.AddLine, CizgiEkle, null, DrawingModeDef.DM_CUSTOM_LINE, (int)IconDef.MI_CURSOR_ARROW);
}

```

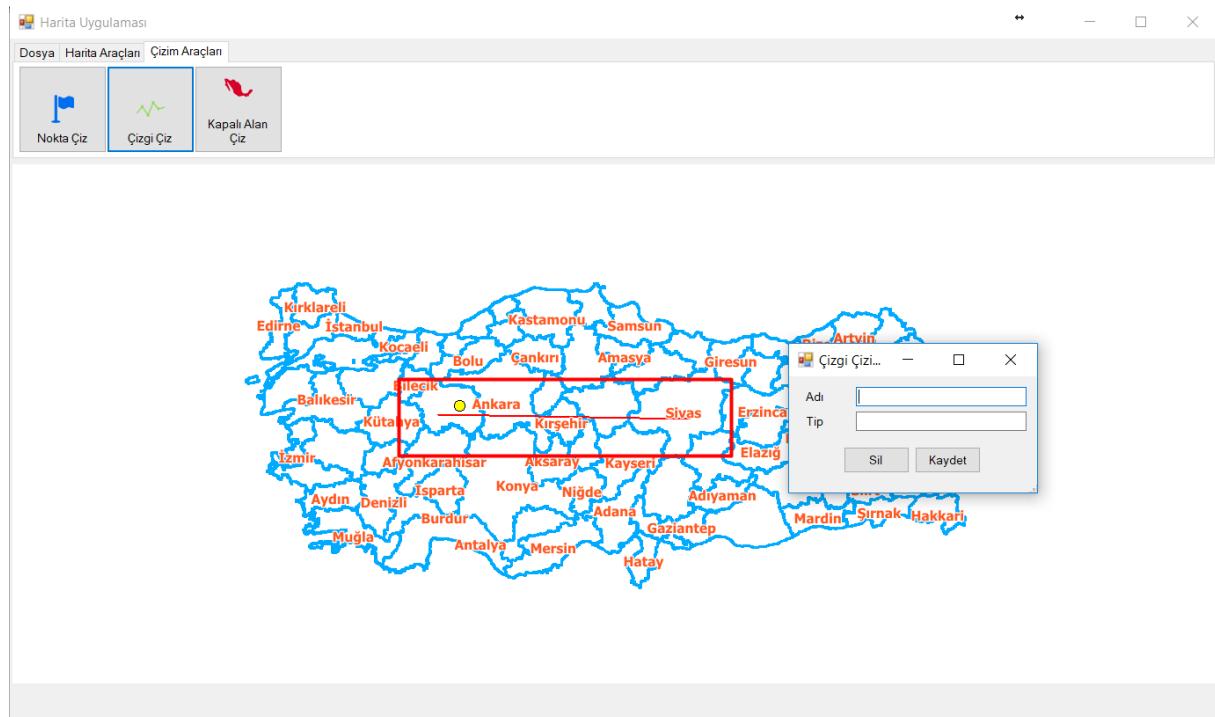
Son olarak ta çizgi çizim aracımızın aktifleştirilmesi için gerekli olan kodlar Form1.cs içerisinde aşağıdaki resimde olduğu gibi eklenmiştir.

```

1 reference
private void buttonCizgiCiz_Click(object sender, EventArgs e)
{
    CustomToolButtons.Instance.ActivateCustomTool(ButtonName.AddLine);
}

```

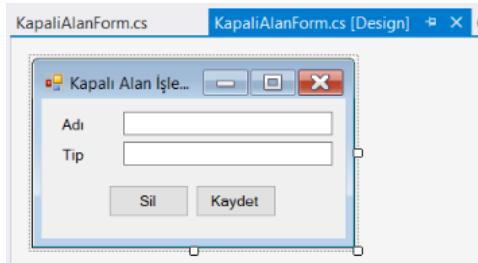
Uygulamamızı çalıştırduğumuzda örnek çizgi çizim aracının ekran görüntüsü aşağıdaki gibi olacaktır.



Kapalı Alan Çizim Aracı

Kapalı Alan çizim aracının geliştirmesi de nokta çizim aracının geliştirmesinin aynısı olarak yapılmaktadır. Burada tek bir farklı nokta bulunmaktadır. Bu nokta Nokta çizimde kullanılan “`commandinfo(1)`” ve “`commandinfo(2)`” metodları yerine direk olarak “`commandinfo(1)`” ile objenin kendisi alınarak “`o`” objesine atanmaktadır. Geri kalan bütün geliştirme adımları aynıdır. Aşağıda sırasıyla kodlar yer almaktadır.

Kapalı Alan çizim işlemleri için aşağıdaki şekilde bir form tasarımlı oluşturulmuştur.



Ardından çizim işlemi tamamlandıktan sonra MapInfo tarafından çalıştırılacak metodumuz “CustomToolButtons” sınıfı içerisinde aşağıdaki gibi oluşturulmuştur.

```
//<summary> Kapalı Alan çizim işleminde. Harita Üzerine kapalı alan çizimi tamamlandıktan sonra MapInfo tarafından çalışan ...</summary>
public void KapaliAlanEkle(object param)
{
    // Yeni bir obje tanımlıyoruz.
    Program.miApplication.RunMapBasicCommand("dim o as object");
    // Tanımlanan objeye yapılan çizimin obje değeri atanıyor
    Program.miApplication.RunMapBasicCommand("o:=commandinfo(1)");
    //Objenin stil bilgisi kaydediliyor.
    Program.miApplication.RunMapBasicCommand("alter object o info 3, makebrush(46, 11796288,-1)");
    // Oluşturulan obje katmana insert ediliyor.
    Program.miApplication.RunMapBasicCommand("insert into KAPALI_ALAN_KATMANI(obj) Values(o)");
    // Eklenen son kayıt tablodan seçiliyor.
    Program.miApplication.RunMapBasicCommand("fetch last from KAPALI_ALAN_KATMANI");
    // Seçilen kaydın rowid bilgisi alınıyor
    var rowId = Program.miApplication.EvalMapBasicCommand("KAPALI_ALAN_KATMANI.rowid");
    // Alınan rowid bilgisi ile tablodan kayıt seçilierek geçici tablolaya aktarılıyor
    Program.miApplication.RunMapBasicCommand("Select * from KAPALI_ALAN_KATMANI where rowid=" + rowId + " into Sel_Alان neselect");
    // Tip ve Adı bilgilerinin girilmesi için Kapalı Alan formu u oluşturuluyor.
    KapaliAlanForm form = new KapaliAlanForm();

    // Bu form aynı şekilde güncelleme işleminde de kullanılacağı için form içerisinde Global bir değişken oluşturularak bu değişkenin değeri false olarak ayarlanıyor.
    form.IsUpdate = false;

    // Nokta formumuzu açıyor.
    DialogResult res = form.ShowDialog();

    // Eğer kullanıcı kaydet butonu dışında formu direk kapatırsa geçici olarak açılan tablo kapatılıyor.
    if (res != DialogResult.OK)
    {
        Program.miApplication.RunMapBasicCommand("rollback table Sel_Alan");
        Program.miApplication.RunMapBasicCommand("close table Sel_Alan");
    }
    // Yukarıda oluşturulan obje kaldırılıyor.
    Program.miApplication.RunMapBasicCommand("undim o");
}
```

Ardından kapalı alan formunda yer alan Kaydet butonun Click metodu içerisinde aşağıdaki resimde görüldüğü gibi oluşturulmuştur.

```

7  using System.Text;
8  using System.Threading.Tasks;
9  using System.Windows.Forms;
10
11 namespace DefaultMix64
12 {
13     public partial class KapaliAlanForm : Form
14     {
15         public bool IsUpdate { get; set; }
16         public KapaliAlanForm()
17         {
18             InitializeComponent();
19         }
20
21         private void KapaliAlanForm_Load(object sender, EventArgs e)
22         {
23         }
24
25         private void buttonKaydet_Click(object sender, EventArgs e)
26         {
27             if (textBoxAdi.Text.Trim().Length <= 0)
28             {
29                 MessageBox.Show("Lütfen Adı bilgisini giriniz.");
30                 return;
31             }
32             if (textBoxTip.Text.Trim().Length <= 0)
33             {
34                 MessageBox.Show("Lütfen Tip bilgisini giriniz.");
35             }
36             // Adı ve Tip bilgileri güncellenmeyecektir
37             Program.miApplication.RunMapBasicCommand("Update Sel_Alın set ADI=\"" + textBoxAdi.Text + "\", TIP=\"" + textBoxTip.Text + "\"");
38             // Güncellenen verileri kaydediyoruz
39             Program.miApplication.RunMapBasicCommand("commit table Sel_Alın");
40             // Açılığımız geçici tabloyu kapatıyoruz
41             Program.miApplication.RunMapBasicCommand("close table Sel_Alın");
42             MessageBox.Show("Kayıt işlemi başarılı bir şekilde yapılmıştır.");
43             this.DialogResult = DialogResult.OK;
44             this.Close();
45         }
46     }
47 }
48

```

Ardından kapalı alan çizim aracıımız aşağıdaki gibi tanımlanmıştır.

```

/// <summary> Özelleştirilmiş araçları yüklemek için kullanılmaktadır.
1 reference
public void LoadCustomTools(System.Windows.Forms.Form mainForm)
{
    this._mainForm = mainForm;
    LoadCustomToolButton(ButtonName.GetInfo, GetInfo, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_HELP);
    LoadCustomToolButton(ButtonName.AddPoint, NoktaEkle, null, DrawingModeDef.DM_CUSTOM_POINT, (int)IconDef.MI_CURSOR_ARROW);
    LoadCustomToolButton(ButtonName.AddLine_Cizikle, null, DrawingModeDef.DM_CUSTOM_LINE, (int)IconDef.MI_CURSOR_ARROW);
    LoadCustomToolButton(ButtonName.AddPolygon, KapaliAlanEkle, null, DrawingModeDef.DM_CUSTOM_POLYGON, (int)IconDef.MI_CURSOR_ARROW);
}

```

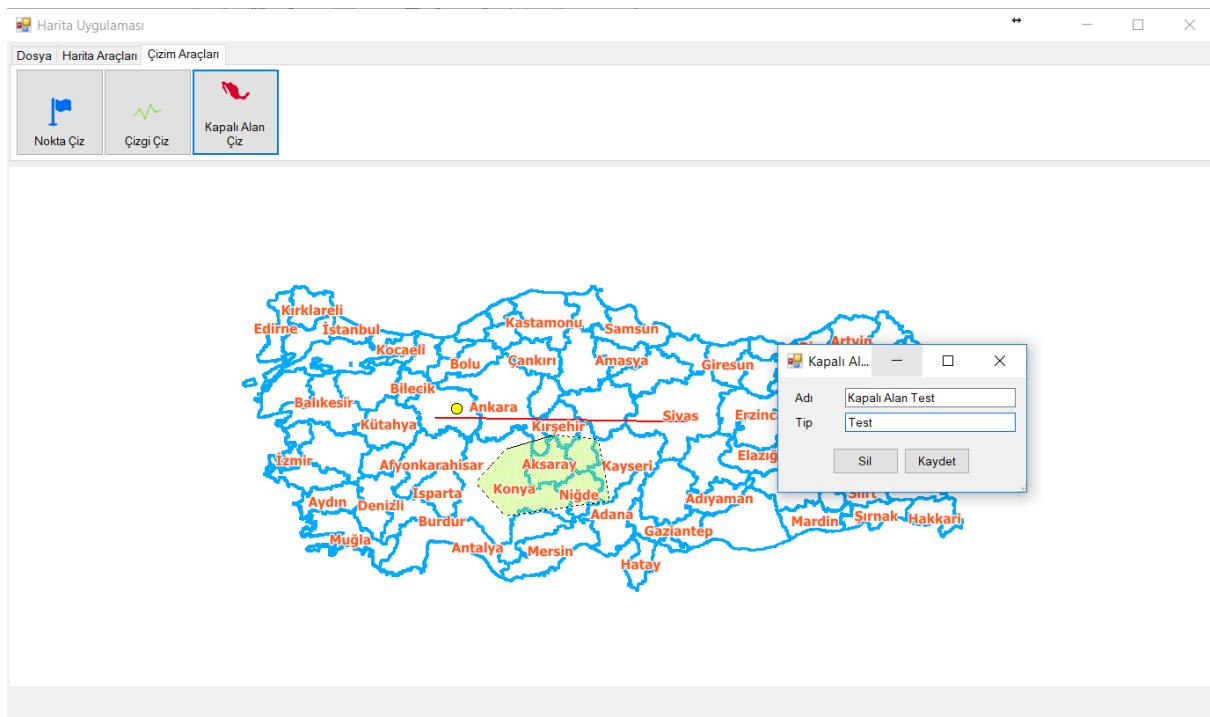
Son olarak ta kapalı alan çizim aracıının aktifleştirilmesi için gerekli olan kodlar Form1.cs içerisinde aşağıdaki resimde olduğu gibi eklenmiştir.

```

1 reference
private void buttonKapaliAlanCiz_Click(object sender, EventArgs e)
{
    CustomToolButtons.Instance.ActivateCustomTool(ButtonName.AddPolygon);
}

```

Uygulamamızı çalıştırduğumda örnek çizgi çizim aracının ekran görüntüsü aşağıdaki gibi olacaktır.



Güncelleme ve Silme İşlevlerinin Geliştirilmesi

Güncelleme ve Silme işlemleri için bilgi al aracı kullanılacaktır. Bu nedenle bilgi al aracında kodlarımızın büyük bir kısmını yazmıştık. Şimdi "Form1.cs" içerisindeki "ShowInfo" metodumuza geri dönerek gerekli kodlamızı yazmaya başlayabiliriz.

Nokta Katmanı Güncelleme ve Silme İşlevi

Nokta katmanımız için veri girişini sağlayan formumuzu daha öncesinde oluşturmuştu. Bu sefer aynı form üzerinde güncelleme ve silme işlemlerini gerçekleştireceğiz. Bunun için öncelikle "Form1.cs" içerisindeki "ShowInfo" metodunda yer alan "**NOKTA_KATMANI**" case 'ine aşağıdaki kodlamızı yazıyoruz. Burada gelen rowid ve tabloadi bilgisi ile tablomuzdan ilgili kaydı seçip geçici tabloya atıyoruz. Daha sonrasında NoktaForm formumuzdaki IsUpdate değişkeninin değerini true yaparak formumuzu açıyoruz.

```

public void ShowInfo(string key)
{
    if (key.Split('#').Length == 2)
    {
        // Bilgisi alınacak kaydın olduğu tablonun ad bilgisi
        string tabloAdi = key.Split('#')[0];

        // Bilgisi alınacak kaydın tablodaki rowid bilgisi
        string rowId = key.Split('#')[1];

        switch (key)
        {
            case "IL":
                [il Bilgisi Alma]
                break;
            case "ILCE":
                [ilce Bilgisi Alma]
                break;
            case "NOKTA_KATMANI":
                #region Nokta Katmanı Bilgi Alma
                // Tablomuzdan bilgisi alınmak istenen kaydı alıp geçici tabloya aktarıyoruz.
                Program.miApplication.RunMapBasicCommand("Select * from " + tabloAdi + " where rowid=" + rowId + " into Sel_Nokta");
                // Geçici tablodaki kaydımızın bilgilere erişebilmek için seçili hale getiriyoruz.
                Program.miApplication.RunMapBasicCommand("fetch first from Sel_Nokta");

                NoktaForm form = new NoktaForm();
                // Güncelleme işlemimizi yaptığımizi belirtmek için değişkenin değerini true olarak setliyoruz.
                form.IsUpdate = true;

                DialogResult res = form.ShowDialog();
                // Eğer kullanıcı silme veya güncelleme işlemi yapmadan formu kapatır ise geçici tablomuzu kapatıyoruz.
                if (res != DialogResult.OK)
                {
                    Program.miApplication.RunMapBasicCommand("rollback table Sel_Nokta");
                    Program.miApplication.RunMapBasicCommand("close table Sel_Nokta");
                }
                #endregion
                break;
            case "CIZGI_KATMANI":
                [Cizgi Katmanı Bilgi Alma]
                break;
            case "KAPALI_ALAN_KATMANI":
                [Kapalı Alan Katmanı Bilgi Alma]
                break;
        }
    }
}

```

Daha sonrasında NoktaForm.cs içerisindeki “NoktaForm_Load” metodumuzun içeriğini aşağıdaki şekilde yazıyoruz.

```

1 reference
private void NoktaForm_Load(object sender, EventArgs e)
{
    if (IsUpdate)
    {
        textBoxAdi.Text = Program.miApplication.EvalMapBasicCommand("Sel_Nokta.ADI");
        textBoxTip.Text = Program.miApplication.EvalMapBasicCommand("Sel_Nokta.TIP");
        // Eğer form güncelleme işlemi için açıldı ise silme butonunu kullanıcı erişimine kapatıyoruz.
        buttonSil.Enabled = true;
    }
    else
    {
        // Eğer form çizim işlemi için açıldı ise silme butonunu kullanıcı erişimine kapatıyoruz.
        buttonSil.Enabled = false;
    }
}

```

Not: Kaydın güncellenmesi için “Kaydet” butonu kullanılacaktır. Bu nedenle ilk çizim işleminde kullanılan tablo adı ile güncelleme işlemindeki geçici tablo isimleri aynı verilmiştir. Bu şekilde ekstra kod yazmadan aynı kod bloğu ile hem kaydetme hemde güncelleme işlemleri yapılabilecektir.

Güncelleme işleminde de aynı “Kaydet” butonu kullanılacağı için “buttonKaydet_Click” metodu üzerinde herhangi bir değişiklik yapılmamıştır.

Silme işlemi için eklenen buttonSil butonumuzun “buttonSil_Click” metodumuza kodlarımızı aşağıdaki şekilde ekliyoruz.

```

1 reference
private void buttonSil_Click(object sender, EventArgs e)
{
    // Kaydı silmek isteyip istemediği bilgisi kullanıcıya sorulmaktadır.
    DialogResult res = MessageBox.Show("Kaydi Silmek İstediğinize Emin Misiniz?", "Uyarı", MessageBoxButtons.YesNo);
    if (res == DialogResult.Yes)
    {
        // Geçici tablomuzdan kaydımızı siliyoruz.
        Program.miApplication.RunMapBasicCommand("delete from Sel_Nokta");

        // Geçici tablomuzu kaydediyoruz.
        Program.miApplication.RunMapBasicCommand("commit table Sel_Nokta");

        // Geçici tablomuzu kapatıyoruz.
        Program.miApplication.RunMapBasicCommand("close table Sel_Nokta");
        MessageBox.Show("Kayıt başarılı bir şekilde silinmiştir.");
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}

```

Çizgi Katmanı Güncelleme ve Silme İşlevi

Çizgi katmanımız için veri girişini sağlayan formumuzu daha öncesinde oluşturmuştu. Bu sefer aynı form üzerinde güncelleme ve silme işlemlerini gerçekleştireceğiz. Bunun için öncelikle “Form1.cs” içerisindeki “ShowInfo” metodunda yer alan “**CIZGI_KATMANI**” case’ine aşağıdaki kodlarımızı yazıyoruz. Burada gelen rowid ve tabloadi bilgisi ile tablomuzdan ilgili kaydı seçip geçici tabloya atıyoruz. Daha sonrasında NoktaForm formumuzdaki IsUpdate değişkeninin değerini true yaparak formumuzu açıyoruz.

```

public void ShowInfo(string key)
{
    if (key.Split('#').Length == 2)
    {
        // Bilgisi alınacak kaydın olduğu tablonun ad bilgisi
        string tabloAdi = key.Split('#')[0];

        // Bilgisi alınacak kaydın tablodaki rowid bilgisi
        string rowId = key.Split('#')[1];

        switch (tabloAdi)
        {
            case "IL":
                [il Bilgisi Alma]
                break;
            case "ILCE":
                [İlçe Bilgisi Alma]
                break;
            case "NOKTA_KATMANI":
                [Nokta Katmanı Bilgi Alma]
                break;
            case "CIZGI_KATMANI":
                #region Çizgi Katmanı Bilgi Alma
                // Tablomuzdan bilgisi alınmak istenilen kaydı alıp geçici tabloya aktarıyoruz.
                Program.miApplication.RunMapBasicCommand("Select * from " + tabloAdi + " where rowid=" + rowId + " into Sel_Cizgi");
                // Geçici tablodaki kaydımızın bilgilere erişebilmek için seçili hale getiriyoruz.
                Program.miApplication.RunMapBasicCommand("fetch first from Sel_Cizgi");

                CizgiForm cizgiform = new CizgiForm();
                // Güncelleme işlemimizi yaptığımizi belirtmek için değişkenin değerini true olarak setliyoruz.
                cizgiform.IsUpdate = true;

                DialogResult resCizgi = cizgiform.ShowDialog();
                // Eğer kullanıcı silme veya güncelleme işlemi yapmadan formu kapatır ise geçici tablomuzu kapatıyor.
                if (resCizgi != DialogResult.OK)
                {
                    Program.miApplication.RunMapBasicCommand("rollback table Sel_Cizgi");
                    Program.miApplication.RunMapBasicCommand("close table Sel_Cizgi");
                }

                #endregion
                break;
            case "KAPALI_ALAN_KATMANI":
                [Kapalı Alan Katmanı Bilgi Alma]
                break;
        }
    }
}

```

Daha sonrasında CizgiForm.cs içerisindeki “CizgiForm_Load” metodumuzun içeriğini aşağıdaki şekilde yazıyoruz.

```
1 reference
private void CizgiForm_Load(object sender, EventArgs e)
{
    if (IsUpdate)
    {
        textBoxAdi.Text = Program.miApplication.EvalMapBasicCommand("Sel_Cizgi.ADI");
        textBoxTip.Text = Program.miApplication.EvalMapBasicCommand("Sel_Cizgi.TIP");
        // Eğer form güncelleme işlemi için açıldı ise silme butonunu kullanıcı erişimine kapatıyoruz.
        buttonSil.Enabled = true;
    }
    else
    {
        // Eğer form çizim işlemi için açıldı ise silme butonunu kullanıcı erişimine kapatıyoruz.
        buttonSil.Enabled = false;
    }
}
```

Not: Kaydın güncellenmesi için “Kaydet” butonu kullanılacaktır. Bu nedenle ilk çizim işleminde kullanılan tablo adı ile güncelleme işlemindeki geçici tablo isimleri aynı verilmiştir. Bu şekilde ekstra kod yazmadan aynı kod bloğu ile hem kaydetme hemde güncelleme işlemleri yapılabilecektir.

Güncelleme işleminde de aynı “Kaydet” butonu kullanılacağı için “buttonKaydet_Click” metodu üzerinde herhangi bir değişiklik yapılmamıştır.

Silme işlemi için eklenen buttonSil butonumuzun “buttonSil_Click” metodumuza kodlarımızı aşağıdaki şekilde ekliyoruz.

```
1 reference
private void buttonSil_Click(object sender, EventArgs e)
{
    // Kaydı silmek isteyip istemediği bilgisi kullanıcıya sorulmaktadır.
    DialogResult res = MessageBox.Show("Kaydı Silmek İstediğinize Emin Misiniz?", "Uyarı", MessageBoxButtons.YesNo);
    if (res == DialogResult.Yes)
    {
        // Geçici tablomuzdan kaydımızı siliyoruz.
        Program.miApplication.RunMapBasicCommand("delete from Sel_Cizgi");

        // Geçici tablомуzu kaydediyoruz.
        Program.miApplication.RunMapBasicCommand("commit table Sel_Cizgi");

        // Geçici tablomuzu kapatıyoruz.
        Program.miApplication.RunMapBasicCommand("close table Sel_Cizgi");
        MessageBox.Show("Kayıt başarılı bir şekilde silinmiştir.");
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

Kapalı Alan Katmanı Güncelleme ve Silme İşlevi

Kapalı Alan katmanımız için veri girişini sağlayan formumuzu daha öncesinde oluşturmuştu. Bu sefer aynı form üzerinde güncelleme ve silme işlemlerini gerçekleştireceğiz. Bunun için öncelikle “Form1.cs” içerisindeki “ShowInfo” metodunda yer alan “**KAPALI_ALAN_KATMANI**” case ‘ine aşağıdaki kodlarımızı yazıyoruz. Burada gelen rowid ve tabloadi bilgisi ile tablomuzdan ilgili kaydı seçip geçici tabloya atıyoruz. Daha sonrasında NoktaForm formumuzdaki IsUpdate değişkeninin değerini true yaparak formumuzu açıyoruz.

```

public void ShowInfo(string key)
{
    if (key.Split('#').Length == 2)
    {
        // Bilgisi alınacak kaydın olduğu tablonun adı bilgisi
        string tabloAdı = key.Split('#')[0];

        // Bilgisi alınacak kaydın tablodaki rowid bilgisi
        string rowId = key.Split('#')[1];

        switch (tabloAdı)
        {
            case "IL":
                İl Bilgisi Alma
                break;
            case "ILCE":
                İlçe Bilgisi Alma
                break;
            case "NOKTA_KATMANI":
                Nokta Katmanı Bilgi Alma
                break;
            case "CIZGI_KATMANI":
                Çizgi Katmanı Bilgi Alma
                break;
            case "KAPALI_ALAN_KATMANI":
                #region Kapalı Alan Katmanı Bilgi Alma
                // Tablomuzdan bilgi alınmak istenen kaydı alıp geçici tabloya aktarıyoruz.
                Program.miApplication.RunMapBasicCommand("Select * from " + tabloAdı + " where rowid=" + rowId + " into Sel_Alın");
                // Geçici tablodaki kaydımızın bilgilere erişebilmek için seçili hale getiriyoruz.
                Program.miApplication.RunMapBasicCommand("fetch first from Sel_Alın");

                KapaliAlanForm alanform = new KapaliAlanForm();
                // Güncelleme işlemini yaptığımizi belirtmek için değişkenin değerini true olarak setliyoruz.
                alanform.IsUpdate = true;
                alanform.ShowDialog();
                // Eğer kullanıcı silme veya güncelleme işlemi yapmadan formu kapatır ise geçici tablomuzu kapatıyoruz.
                if (resAlan != DialogResult.OK)
                {
                    Program.miApplication.RunMapBasicCommand("rollback table Sel_Alın");
                    Program.miApplication.RunMapBasicCommand("close table Sel_Alın");
                }
                #endregion
                break;
        }
    }
}

```

Daha sonrasında KapaliAlanForm.cs içerisindeki “KapaliAlanForm_Load” metodumuzun içeriğini aşağıdaki şekilde yazıyoruz.

```

1 reference
private void KapaliAlanForm_Load(object sender, EventArgs e)
{
    if (IsUpdate)
    {
        textBoxAdı.Text = Program.miApplication.EvalMapBasicCommand("Sel_Nokta.ADI");
        textBoxTip.Text = Program.miApplication.EvalMapBasicCommand("Sel_Nokta.TIP");
        // Eğer form güncelleme işlemi için açıldı ise silme butonunu kullanıcıya erişimine kapatıyoruz.
        buttonSil.Enabled = true;
    }
    else
    {
        // Eğer form çizim işlemi için açıldı ise silme butonunu kullanıcıya erişimine kapatıyoruz.
        buttonSil.Enabled = false;
    }
}

```

Not: Kaydın güncellenmesi için “Kaydet” butonu kullanılacaktır. Bu nedenle ilk çizim işleminde kullanılan tablo adı ile güncelleme işlemindeki geçici tablo isimleri aynı verilmiştir. Bu şekilde ekstra kod yazmadan aynı kod bloğu ile hem kaydetme hemde güncelleme işlemleri yapılabilecektir.

Güncelleme işleminde de aynı “Kaydet” butonu kullanılacağı için “buttonKaydet_Click” metodu üzerinde herhangi bir değişiklik yapılmamıştır.

Silme işlemi için eklenen buttonSil butonumuzun “buttonSil_Click” metodumuza kodlarımızı aşağıdaki şekilde ekliyoruz.

```

1 reference
private void buttonSil_Click(object sender, EventArgs e)
{
    // Kaydı silmek isteyip istemediği bilgisi kullanıcıya sorulmaktadır.
    DialogResult res = MessageBox.Show("Kaydı Silmek İstediğinize Emin Misiniz?", "Uyarı", MessageBoxButtons.YesNo);
    if (res == DialogResult.Yes)
    {
        // Geçici tablomuzdan kaydımızı siliyoruz.
        Program.miApplication.RunMapBasicCommand("delete from Sel_Nokta");

        // Geçici tablomuzu kaydediyoruz.
        Program.miApplication.RunMapBasicCommand("commit table Sel_Nokta");

        // Geçici tablomuzu kapatıyoruz.
        Program.miApplication.RunMapBasicCommand("close table Sel_Nokta");
        MessageBox.Show("Kayıt başarılı bir şekilde silinmiştir.");
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}

```

Genel Notlar

Not 1

MapInfo kütüphanesi arka planda MapBasic dilini kullanmaktadır. Bu nedenle C# ile uygulama geliştirmede MapBasic komutlarını bilmekte fayda vardır. Çünkü C# tarafından MapInfo daki MapBasic komutları çalıştırılmaktadır. Örnek olarak “`commandinfo(1)`”, “`makepen`”, “`fetch last from CIZGI_KATMANI`” komutları verilebilir.

Not 2

MapInfo case-sensitive çalışmamaktadır. Yani MapBasic te oluşturduğunuz bir değişken ismi veya yapılan bir sorgu sonucunu attığınız geçici tablonun ismini büyük veya küçük harf yazabilirsiniz. Örnek olarak “`o`” olarak obje değişkenine “`O`” olarakca erişim sağlayabilirsiniz. Yine geçici tabloya verilen “`Sel_Cizgi`” ismine “`SEL_CIZGI`” veya “`sel_cizgi`” şeklinde erişim sağlanabilmektedir.