Program.cs :-

* Inside program.cs file Main method are available.
* Means start Running program from here

Inside main method create one networkInterface[] array

* Provides configuration and statistical information for a network interface.
* This class encapsulates data for network interfaces, also known as adapters, on the local computer. You do not create instances of this class; the [GetAllNetworkInterfaces](https://docs.microsoft.com/en-us/dotnet/api/system.net.networkinformation.networkinterface.getallnetworkinterfaces?view=net-6.0) method returns an array that contains one instance of this class for each network interface on the local computer.

Then create two variable one is string and second is Boolean.

* String sMacAddress = string.Empty; //set string as empty
* bool isGeniuene = false; // isGeniuene value is false

IPInterfaceProperties:-

* C# IPInterfaceProperties type Provides information about network interfaces that support Internet Protocol version 4 (IPv4) or Internet Protocol version 6 (IPv6).

sMacAddress = adapter.GetPhysicalAddress().ToString();

* Get mac address and store inside sMacAddress.
* Mac address store in string format.

After that check mac address is match or not using if.

* mac address is match then set isGeniuene value true.

After that check

* if(isGeniuene)//condition is true then go inside this if block {}

else

{

MessageBox.Show("Please insert License file to proceed...!");  
}

inside if create one local variable bool mutexCreated = true;

Mutex

* A Mutex is a synchronization primitive that can also be used for interprocess synchronization. When two or more threads need to access a shared resource at the same time, the system needs a synchronization mechanism to ensure that only one thread at a time uses the resource. Mutex is a synchronization primitive that grants exclusive access to the shared resource to only one thread. If a thread acquires a Mutex, the second thread that wants to acquire that Mutex is suspended until the first thread releases the Mutex.
* In short, A mutual exclusion ("Mutex") is a mechanism that acts as a flag to prevent two threads from performing one or more actions simultaneously. The entire action that you want to run exclusively is called a critical section or protected section.
* A critical section is a piece of code that accesses a shared resource (data structure or device) but the condition is that only one thread can enter in this section at a time.

After that check condition

if(mutexCreated) //true

{

Application.Run(new MasterForm()); //call MasterForm()

}

**MasterFrom.cs :-**

* Inside MasterFrom.cs file all aubut the masterfrom backend codding work.

**InitializeComponent();**

* This method call the designer class
* Example:

**MasterFrom.Designer.cs**

After that call **CheckHalconLicense();** method and check condition

If condition is true then return true otherwise return false.

And return value store inside HalconLicense Boolean variable.

* if (HalconLicense) //true
* Inside if

hSmartWindowControl1.HKeepAspectRatio = true;

hSmartWindowControl2.HKeepAspectRatio = true;

win\_result1 = hSmartWindowControl1.HalconWindow;

win\_result2 = hSmartWindowControl2.HalconWindow;

// hSmartWindowControl1.SetFullImagePart(null);

// hSmartWindowControl2.SetFullImagePart(null);

Cam1ResultDict = new ConcurrentDictionary<int, bool>();

Cam2ResultDict = new ConcurrentDictionary<int, bool>();

FinalResultDict = new ConcurrentDictionary<int, bool>();

ReadCalibration\_files();

* **After that check camera used using if else statement.**

if (usedcamera == "IDS")

{

Cam2 = initCamera(Cam2, 1); // call initCamera() method.

if (Cam2 != null) //check condition

{

Cam2.EventFrame += new System.EventHandler(this.onFrameEvent\_cam2);

// call onFrameEvent\_cam2() method

}

}

else if (usedcamera == "FLIR")

{

InitImageEventListener(); //call this method

}

else if (usedcamera == "HIKVISION")

{

initcommunication(); //call initcommunication() method

DeviceListAcq(); // call DeviceListAcq()

Init(); // call Init() method

}

**Inside Init() Method**

protected void Init()

{

try

{

if (0 == stDevList.nDeviceNum) //check condition filter

{

return;

}

Open(); //call open method

}

catch (NullReferenceException ex)

{

MessageBox.Show("Camera initialization failed.Please check Camera connection." + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("Camera initialization failed.Please check Camera connection." + ex.Message);

}

}

**Inside Open() Method**

nRet = device1.MV\_CC\_OpenDevice\_NET(); //device1 is a camera class object using this object call MV\_CC\_OpenDevice\_NET() method and store return value inside nRet variable.

**Exception :**

catch (NullReferenceException e)

{

MessageBox.Show("open error\n" + e.Message);

}

catch (SystemException ex)

{

MessageBox.Show("Camera initialization failed Please check Camera1 connection." + ex.Message);

}

//same process follow to set device2 camera

**//From Load**

private void MasterForm\_Load(object sender, EventArgs e)

{

try

{

this.WindowState = FormWindowState.Maximized; //display full scrn

this.Location = new Point(0, 0);// set distance first is X and second is Y. X is used of distance from left, Y is distance from upper side.

this.Size = Screen.PrimaryScreen.WorkingArea.Size; // set screen according to working pc or laptop screen size.

PPM\_timer = new System.Timers.Timer(10000); //set 10 second time

PPM\_timer.Elapsed += OnTimedEvent; // call every 10 second OnTimeEvent Method.

PPM\_timer.AutoReset = true;//run timer for only once.

PPM\_timer.Start(); //start timer

Login\_Data\_Ob = LoginData.getInstance(); //get instance

lbl\_user\_name.Text = Login\_Data\_Ob.User\_Name; // set username text.

isInspection = 0;

}

catch (NullReferenceException ex)

{

MessageBox.Show("m form loading error" + ex.Message);

}

catch (Exception ex)

{

MessageBox.Show("m form loading error" + ex.Message);

}

}

**MV\_CC\_DEVICE\_INFO\_LIST:**

* MV\_CC\_DEVICE\_INFO\_LIST is the one of the structure inside that structure store one unsigned integer and one IntPtr [] pointer/ reference. **stDevList** is the instance of **MV\_CC\_DEVICE\_INFO\_LIST** structure.
* **Return value store inside nRet. nRet is the integer variable.**

**Marshal class:**

* Provides a collection of methods for allocating unmanaged memory, copying unmanaged memory blocks, and converting managed to unmanaged types, as well as other miscellaneous methods used when interacting with unmanaged code.