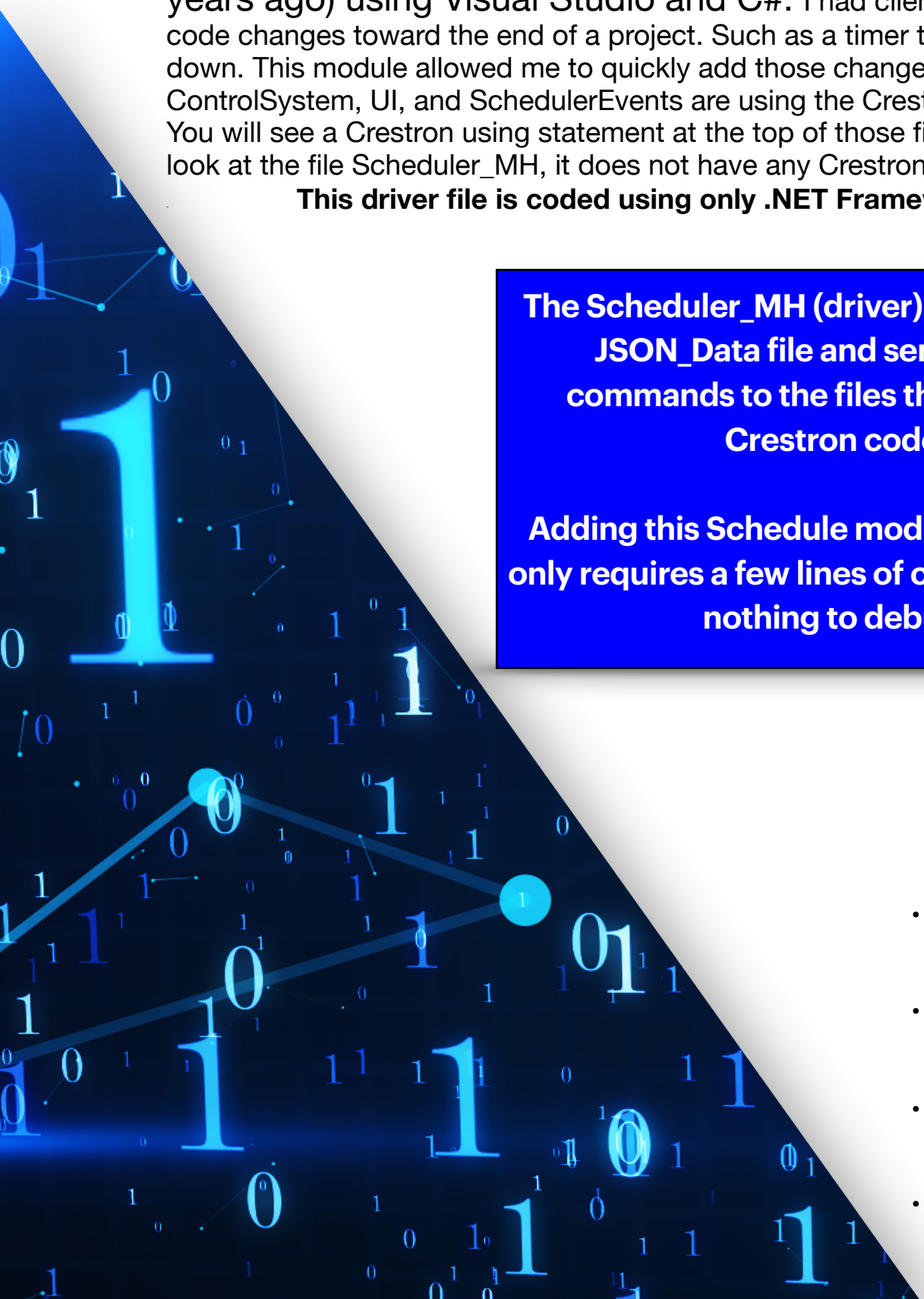
SCHEDULER - CRESTRON & C# CODE EXAMPLE

This is an example of a custom module that I developed (few years ago) using Visual Studio and C#. I had clients that asked for code changes toward the end of a project. Such as a timer to shut the system down. This module allowed me to quickly add those changes. The file ControlSystem, UI, and SchedulerEvents are using the Crestron C# framework. You will see a Crestron using statement at the top of those files. But when you look at the file Scheduler_MH, it does not have any Crestron using statements.

This driver file is coded using only .NET Framework.



The Scheduler_MH (driver) file creates the JSON_Data file and send/receive commands to the files that are using Crestron code.

Adding this Schedule module to a project only requires a few lines of code, and there's nothing to debug.

Attached Files:

- ControlSystem (**Crestron C# code**)
- UI (**Crestron C# code**)
- SchedulerEvents (**Crestron C# code**)
- Scheduler_MH (**Microsoft / C# only**)

When developing modules for equipment, develop the module in a standard language. Then the module is not dependent on Crestron, AMX, or any manufacturer.

THERE IS A LINK IN THE COMMENTS SECTION TO A VIDEO SHOWING HOW THIS CODE WORKS.

In this video you can see how the custom Scheduler module works. A company could give the files (source code) that has Crestron code in it but only provide the DLL file that does the work — in this example, the file that edits the JSON data — Scheduler_MH. **This approach gives the customers the flexibility to support the system in the future and protect the company IP.**

SCHEDULER_MH

```
using System;
using Newtonsoft.Json;
using System.Text;
using System.Collections.Generic;
```

```
/// <summary>
// Dynamic Link Library
//
/// </summary>
///
namespace Scheduler_MH
{
    public class Scheduler
    {
        #region Events and Delegates
        public delegate void DeleteAnalog(bool DeleteAnalog);
        public event DeleteAnalog s1_DeleteAnalogEvent;
        public delegate void EditAnalog(bool EditAnalog);
        public event EditAnalog s1_EditAnalogEvent;
        public delegate void ListMaskSubPage(bool AddAnalog);
        public event ListMaskSubPage s1_ListMaskSubPageEvent;
        public delegate void DeleteMaskSubPage(bool AddAnalog);
        public event DeleteMaskSubPage s1_DeleteMaskSubPageEvent;
```

```
        public delegate void Info(NewData data);
        public event Info s1_InfoEvent;
        public delegate void ScheduleSave(bool Save);
        public event ScheduleSave s1_SaveEvent;
        public delegate void SaveingSchedule(bool data);
        public event SaveingSchedule s1_SaveingEvent;
        #endregion
        #region Propertys
        private CCriticalSection myCCriticalSectionSO = new
CCriticalSection();
        private CCriticalSection myCCriticalSectionPush = new
CCriticalSection();
        public static List<Scheduler> allScheduler = new
List<Scheduler>();
        public ScheduleJason Schedule;
        private static CTimer internalTimer, internalTimer2;
        public string _DaString;
        string _FileName { get; set; }
        public Action<List<string>> s1_EventNames;
        NewData AddEditSchedule;
```

```
        private Thread SchedulerThread_1, SchedulerThread_2;
        private int SaveItem, DeleteItem, EditItem, AddItem,
Hr_SmartObject, Min_SmartObject,
Day_SmartObject,ScheduleList_SmartObject,
            SelectScheduleItem, Hr, Min,Task_SmartObject;
        private bool setHr_SmartObject, setMin_SmartObject,
setTask_SmartObject, setDay_SmartObject, _Delete, _Edit;
        #endregion
        #region Constructor
        public Scheduler(string name,int Delete, int Edit, int Add,
int _Hr_SmartObject, int _Min_SmartObject,int _Day_SmartObject,
            int _ScheduleList_SmartObject, int _Task_SmartObject, int
_SaveItem) {
try {
_FileName = name;
DeleteItem = Delete;
EditItem = Edit;
AddItem = Add;
Hr_SmartObject = _Hr_SmartObject;
Min_SmartObject =_Min_SmartObject;
```

```
Day_SmartObject = _Day_SmartObject;

ScheduleList_SmartObject = _ScheduleList_SmartObject;

Task_SmartObject = _Task_SmartObject;

SaveItem = _SaveItem;

AddEditSchedule = new NewData();

            internalTimer = new CTimer(CB, null, 9000, 1000);

          //SchedulerThread_1 = new Thread(SmartObjectCB, null,

Thread.eThreadStartOptions.Running);

         // SchedulerThread_2 = new Thread(PushCB, null,

Thread.eThreadStartOptions.Running);

           // SchedulerThread_2 = new Thread(PushCB, _SelectItem,

Thread.eThreadStartOptions.Running);

            string _driverFilename = string.Format("{0}\\Extensions\

\" + "schedule_3.json", Directory.GetApplicationDirectory());

          //string _driverFilename = string.Format("{0}\\" +

"schedule_3.json", Directory.GetApplicationDirectory());

          //JBuilderConfig getStartData = new

JBuilderConfig(string.Format("{0}\\schedule_3.json",

Directory.GetApplicationDirectory()), 1);

          InitializeFile(_driverFilename);
```

```
            allScheduler.Add(this);
                Helper.PrintLine("Constructor_2:");
            }
            catch (Exception e)
            {
                Helper.PrintLine("Scheduler Driver - Error in the
constructor: {0}", e.Message);
} }
        #endregion
        #region Methods
        public void InitializeFile(string _FilePath)
        {
try {
                if
(Crestron.SimplSharp.CrestronIO.File.Exists(_FilePath))
sure the file is their
                {
// make
                    StreamReader daFile = new StreamReader(_FilePath);
                    _DaString = daFile.ReadToEnd();
```

```
                daFile.Close();
} else {
File Not found");
            }
Helper.PrintLine("Scheduler Driver - Configuration
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Error Reader:
{0}", e.Message);
} }
      public void CB(object obj)
      {
try {
if (_DaString != null)
{
   setDeserializeObject(_DaString);
   internalTimer.Stop();
1000);
e.Message);
```

```
        internalTimer2 = new CTimer(CB2, null, 4000,
} else
        Helper.PrintLine("Object Null");
    internalTimer.Stop();
}
catch (Exception e)
{
    Helper.PrintLine("Scheduler Driver - Error CB - ",
    internalTimer.Stop();
}
e.Message);
}
public void CB2(object obj)
{
try {
        Scheduler.allScheduler[0].getScheduleList();
    }
    catch (Exception e)
    {
        Helper.PrintLine("Scheduler Driver - Error CB2 - ",
```

```
        internalTimer2.Stop();
    }
internalTimer2.Stop();
        }
        public void setDeserializeObject(string data)
        {
try {
            Schedule =
JsonConvert.DeserializeObject<ScheduleJason>(data);
            Helper.PrintLine("setDeserializeObject");
          }
          catch (Exception e)
          {
            Helper.PrintLine("Scheduler Driver - Error
setDeserializeObject: {0}", e.Message);
} }
        private void setDelete(int item)
        {
try {
x.Invoke(true));
```

```
s1_DeleteMaskSubPageEvent.IfNotNull(x =>
s1_DeleteAnalogEvent.IfNotNull(x => x.Invoke(false));
s1_EditAnalogEvent.IfNotNull(x => x.Invoke(false));
            _Delete = false;
            _Edit = false;
            setHr_SmartObject = false;
            setMin_SmartObject = false;
            setTask_SmartObject = false;
            setDay_SmartObject = false;
            Schedule.data.Remove(Schedule.data[item]);
            Scheduler_MH.DeleteItem.start(@"\NVRAM\" +_FileName ,
JsonConvert.SerializeObject(Schedule));
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Error in
setDelete: {0}", e.Message);
} }
    #region Static Methods
    public static void DeleteScheduleItem()
```

```
        {
try {
Scheduler.allScheduler[0].s1_DeleteMaskSubPageEvent.IfNotNull(x =>
x.Invoke(false));
            }
            catch (Exception e)
            {
                Helper.PrintLine("Scheduler Driver - Error in
DeleteScheduleItem: {0}", e.Message);
} }
        public static void SavingItem()
        {
try {
                Scheduler.allScheduler[0].s1_SaveingEvent.IfNotNull(x
=> x.Invoke(false));
            }
            catch (Exception e)
            {
                Helper.PrintLine("Scheduler Driver - Error SavingItem
- Count {0}", e.Message);
```

```
} }
        #endregion
        public void getScheduleList()
        {
Scheduler.allScheduler[0].getScheduleList();
d.Name);
try {
}
List<string> allSchedule_List = new List<string>();
foreach (var d in Schedule.data)
{
    allSchedule_List.Add(d.Name);
    Helper.PrintLine("Scheduler Driver - NAME" +
            s1_EventNames.IfNotNull(x =>
x.Invoke(allSchedule_List));
            Helper.PrintLine("Scheduler Driver -
getScheduleList");
        }
        catch (Exception e)
        {
```

```
            CrestronConsole.PrintLine("Scheduler Driver - Error in
getScheduleList: {0}", e.Message);
} }
        private void setSave(int item)
        {
try {
s1_SaveingEvent.IfNotNull(x => x.Invoke(true));
var newScheduleObj = new ListOfScheduleJason()
{
    Sun = AddEditSchedule.Days[0],
    Mon = AddEditSchedule.Days[1],
    Tue = AddEditSchedule.Days[2],
    Wed = AddEditSchedule.Days[3],
    Thu = AddEditSchedule.Days[4],
    Fri = AddEditSchedule.Days[5],
    Sat = AddEditSchedule.Days[6],
    Hr = AddEditSchedule.Hr,
    Min = AddEditSchedule.Min,
    Name = AddEditSchedule.Name,
    SelectTask = AddEditSchedule.SelectTask
```

```
};
  Schedule.data.Add(newScheduleObj);
            SaveFile.start(@"\NVRAM\Schedule.json",
JsonConvert.SerializeObject(Schedule));
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Error in
setSave: {0}", e.Message);
} }
      private void setAdd()
{
try
        {
            _Delete = false;
            _Edit = false;
            s1_DeleteAnalogEvent.IfNotNull(x => x.Invoke(false));
            s1_EditAnalogEvent.IfNotNull(x => x.Invoke(false));
            setHr_SmartObject = false;
            setMin_SmartObject = false;
```

```
            setTask_SmartObject = false;
            setDay_SmartObject = false;
            AddEditSchedule.Days[0] = false;
            AddEditSchedule.Days[1] = false;
            AddEditSchedule.Days[2] = false;
            AddEditSchedule.Days[3] = false;
            AddEditSchedule.Days[4] = false;
            AddEditSchedule.Days[5] = false;
            AddEditSchedule.Days[7] = false;
            AddEditSchedule.Hr = 0;
            AddEditSchedule.Min = 0;
            AddEditSchedule.SelectTask = null;
            AddEditSchedule.Name = "Event_" +
Schedule.data.Count.ToString();
            s1_InfoEvent.IfNotNull(x =>
x.Invoke(AddEditSchedule));
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Error in  setAdd:
```

```
{0}", e.Message);
} }
        private bool getAllTask()
        {
try {
            if((setHr_SmartObject) && (setMin_SmartObject)
&&(setDay_SmartObject) &&(setTask_SmartObject))
                return true;
            else
             return false;
          }
          catch (Exception e)
          {
             Helper.PrintLine("Scheduler Driver - Exception
getAllTask - {0}", e.Message);
             return false;
} }
        #region Input Data
        public void Click(int join)
        {
```

```
try {
            //_SelectItem = join;
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Exception Push
Click - {0}", e.Message);
} }
    public void Click(int id, int join)
    {
try {
        }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - - Exception Smart
Object Click - {0}", e.Message);
} }
    #endregion
    #endregion
    #region CallBack
```

```
        private void PushCB(int _SelectItem)
        {
            //myCCriticalSectionPush.Enter();
            try
            {
                if (_SelectItem == DeleteItem)
                {
                    _Delete = !_Delete;
                    _Edit = false;
                    Helper.PrintLine("PushCB DeleteItem ");
                    s1_DeleteAnalogEvent.IfNotNull(x =>
x.Invoke(_Delete));
                    s1_EditAnalogEvent.IfNotNull(x =>
                }
x.Invoke(false));
PushCB(join);
SmartObjectCB(id, join);
Helper.PrintLine("Scheduler Driver - SO Click");
//_SelectSamrtObject = id;
//_SelectSamrtObjectItem = join;
```

```
x.Invoke(false));

x.Invoke(_Edit));

            }

_Delete = false;

_Edit = !_Edit;

s1_DeleteAnalogEvent.IfNotNull(x =>

s1_EditAnalogEvent.IfNotNull(x =>

else if (_SelectItem == EditItem)

{

            else if (_SelectItem == AddItem)

               setAdd();

            else if (_SelectItem == SaveItem)

               setSave(SelectScheduleItem);

else {
}

            if ((_Delete) || (_Edit))

               s1_ListMaskSubPageEvent.IfNotNull(x =>

x.Invoke(true));

            else

               s1_ListMaskSubPageEvent.IfNotNull(x =>
```

```
            Helper.PrintLine("PushCB");
            Thread.Sleep(5);
//return o; }
        catch (Exception e)
        {
            Helper.PrintLine("Scheduler Driver - Exception push
thread CB - {0}", e.Message);
            //return null;
        }
        //myCCriticalSectionPush.Leave();
//return o; }
    private void SmartObjectCB(int _SelectSamrtObject, int
SelectSamrtObjectItem)
    {
      // myCCriticalSectionSO.Enter();
       Helper.PrintLine("SmartObjectCB " + "_SelectSamrtObject--"
+ _SelectSamrtObject + "SelectSamrtObjectItem -- " +
SelectSamrtObjectItem);
        Helper.PrintLine("TEST -- ScheduleList_SmartObject " +
x.Invoke(false));
```

```
ScheduleList_SmartObject.ToString());
        try
        {
            #region Smart Object Select A Schedule Item
            if (_SelectSamrtObject == ScheduleList_SmartObject)
            {
                Helper.PrintLine("TEST 5- Scheduler
ScheduleList_SmartObject");
1;
if (_Delete)
   setDelete(SelectScheduleItem);
else// edit {
   SelectScheduleItem = SelectSamrtObjectItem -
                AddEditSchedule.Days[0] =
Schedule.data[SelectScheduleItem].Sun;
                AddEditSchedule.Days[1] =
Schedule.data[SelectScheduleItem].Mon;
                AddEditSchedule.Days[2] =
Schedule.data[SelectScheduleItem].Tue;
                AddEditSchedule.Days[3] =
```

```
Schedule.data[SelectScheduleItem].Wed;
                AddEditSchedule.Days[4] =
Schedule.data[SelectScheduleItem].Thu;
                AddEditSchedule.Days[5] =
Schedule.data[SelectScheduleItem].Fri;
                AddEditSchedule.Days[6] =
Schedule.data[SelectScheduleItem].Sat;
                AddEditSchedule.Hr =
Schedule.data[SelectScheduleItem].Hr;
                AddEditSchedule.Min =
Schedule.data[SelectScheduleItem].Min;
                AddEditSchedule.Name =
Schedule.data[SelectScheduleItem].Name;
                AddEditSchedule.SelectTask =
Schedule.data[SelectScheduleItem].SelectTask;
}
            Helper.PrintLine("TEST 10 - Scheduler Driver " +
AddEditSchedule.Name);
            s1_InfoEvent.IfNotNull(x =>
x.Invoke(AddEditSchedule));
```

```
}
        #endregion
        #region Smart Object Schedule Propertys
else {
(_SelectSamrtObject == Min_SmartObject))
            {
                #region Smart Object Hours
                if (_SelectSamrtObject == Hr_SmartObject)
                {
                    switch (_SelectSamrtObject)
                    {
case 1: {
break; }
case 2: {
break; }
} }
                #endregion
                #region Smart Object Minutes
                else
                {
```

```
                    switch (_SelectSamrtObject)
                    {
case 1: {
}
break; }
case 2: {
#region Smart Object Hours or Minutes
setHr_SmartObject = true;
if ((_SelectSamrtObject == Hr_SmartObject) ||
} }
#endregion
if (Hr < 23)
   Hr += 1;
if (Hr > 2)
   Hr -= 1;
if (Min < 59) {
   if (Hr <= 23)
      Min += 1;
   if (Min > 1)
      Min -= 1;
```

```
break; }
                string _Hr, _Min;
                if (AddEditSchedule.Hr > 9)
                    _Hr = AddEditSchedule.Hr.ToString();
                else
                    _Hr = "0" + AddEditSchedule.Hr.ToString();
                if (AddEditSchedule.Min > 9)
                    _Min = Min.ToString();
                else
                    _Min = "0" + Min.ToString();
                AddEditSchedule._Time = _Hr + ":" + _Min;
            }
            #endregion
            #region Smart Object Select Days
            else if (_SelectSamrtObject == Day_SmartObject)
            {
                AddEditSchedule.Days[SelectSamrtObjectItem]
= !AddEditSchedule.Days[SelectSamrtObjectItem];
                setDay_SmartObject = true;
            }
```

```
                #endregion

                #region Smart Object Select A Task

                else if (_SelectSamrtObject == Task_SmartObject)

                {

                    AddEditSchedule.SelectTask =

Schedule.data[SelectScheduleItem - 1].SelectTask;

// }

}

#endregion }

#endregion

//   myCCriticalSectionSO.Leave();

  // Thread.Sleep(4000);

return o;

          catch (Exception e)

          {

             Helper.PrintLine("Scheduler Driver - Exception in

Smart Object CB Section thread - {0}", e.Message);

//

} }

return null;
```

```
public string getName(string _name)

{

try {

var rand = new Random();

bool check = true;

foreach (var item in Schedule.data)

            {

                if (string.Equals(_name, item.Name))

                    check = false;

}
if (check)

                return _name;

            int t = rand.Next();

            return _name + "_" + t.ToString();

        }

        catch (Exception e)

        {

            Helper.PrintLine("Scheduler Driver - Exception in

Smart Object CB Section thread - {0}", e.Message);

            return null;
```

```
        }
    }
    #endregion
    #region Subcleass
    public class NewData
    {
        public bool[] Days;
        public int Hr, Min, TaskCount;
        public string[] Task;
        public string Name,SelectTask,_Time;
        public NewData()
        {
try {
            Days = new bool[] { false, false, false, false,
false, false, false };
Presets" };
    Task = new string[] { "System Off", "Light
    Hr = 0;
    Min = 0;
    Name = " ";
```

```
    SelectTask = " ";
}
catch (Exception e)
{
            Helper.PrintLine("Scheduler Driver - Exception in
Smart Object CB Section thread - {0}", e.Message);
} }
} #endregion
} }
```

SCHEDULER EVENTS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Crestron.SimplSharp;
using UI_Base_Extensions;
using Scheduler_MH;
```

```
namespace ScheduleApp
{
   public static class SchedulerOfficeEvents
   {
      public static void TaskNameEvent(string data)
      {
try {
            UI.set(18, data);
         }
         catch (Exception e)
         {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - TaskNameEvent " + e.Message);
} }
      public static void EventsList(List<string> items)
      {
try {
align=\"right\"> <FONT size=\"22\" face=\"Arial\"
color=\"#ffffff\">{0}</color></FONT>", item);
                UI.setSO(6, Convert.ToUInt32(r), value2);
```

```
r += 1; }
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - EventsList" + e.Message);
} }
    public static void EditEvent(bool data)
    {
CrestronConsole.PrintLine("TEST -- TEST");
int r = 1;
UI.setSO(6, items.Count);
foreach (var item in items)
{
   string value2 = string.Format(" <P
if (data) {
            UI.set(67, true);
            UI.set(2, false);
        }
else {
```

```
            UI.set(2, true);
        }
    }
    public static  void Schedule_Data(Data_Edit DataEdit)
    {
        for (int item = 0; item < 7; item ++)
        {
            int index = item+1;
            UI.setSO(16, (uint)index, DataEdit.Days[item]);
        }
        UI.set(44, DataEdit.Time);
        UI.set(3, DataEdit.Name);
        UI.set(9, DataEdit.SelectTask);
    }
    public static void SubscribesEvents(bool obj)
    {
    }
    public static void EditAnalogEvent(bool EditAnalog)
    {
try {
```

```
            UI.set(6, EditAnalog ? (ushort)1 : (ushort)0);
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - EditAnalogEvent" + e.Message);
} }
    public static void DeleteAnalogEvent(bool DeleteAnalog)
    {
try {
            CrestronConsole.PrintLine("DeleteAnalogEvent " +
DeleteAnalog.ToString());
            UI.set(5, DeleteAnalog ? (ushort)1 : (ushort)0);
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - DeleteAnalogEvent" + e.Message);
UI.set(67, false);
} }
```

```
        public static void SaveingEvent(bool subPage_Saveing)
        {
try {
            UI.set(9, subPage_Saveing);
          }
          catch (Exception e)
          {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - SaveingEvent" + e.Message);
} }
        public static void SaveEvent(bool subPage_SaveButton)
        {
try {
          }
          catch (Exception e)
          {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - SaveEvent" + e.Message);
} }
        public static void DeleteMaskSubPage(bool flag)
```

```
        {
try {
            UI.set(18, flag ? true : false);
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - DeleteMaskSubPage" + e.Message);
} }
        public static void Task(bool flag)
        {
try {
            UI.set(9, flag ? true : false);
        }
if (subPage_SaveButton)
{
   UI.set(2, true);
   UI.set(203, subPage_SaveButton);
}
else
```

```
    UI.set(203, subPage_SaveButton);
            catch (Exception e)
            {
                CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - Task" + e.Message);
} }
        public static void ListMaskSubPageEvent(bool subPage_ListMask)
        {
try {
                UI.set(8, subPage_ListMask ? true : false);
            }
            catch (Exception e)
            {
                CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - SaveEvent" + e.Message);
} }
        public static void DeleteMaskSubPageEvent(bool subPage_Delete)
        {
try {
                UI.set(201, subPage_Delete);
```

```
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Exception -
SchedulerOfficeEvents - SaveEvent" + e.Message);
}}
}}
```

UI

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Crestron.SimplSharp;
using Crestron.SimplSharpPro.UI;
using Crestron.SimplSharpPro;
using Crestron.SimplSharpPro.Diagnostics;
System Monitor Access
```

```
using Crestron.SimplSharpPro.DeviceSupport;
Generic Device Support
using Crestron.SimplSharp.CrestronIO;
using ScheduleApp;
namespace UI_Base_Extensions

{

   public static class UI

   {

      public static void set(uint join, bool value)

      {

try {

            for (int t = 0; t <

ControlSystem.allPanelList.Count(); t++)

{

// For // For

ControlSystem.allPanelList[t].BooleanInput[join].BoolValue = false; //

riseing edge type control

ControlSystem.allPanelList[t].BooleanInput[join].BoolValue = value;

            }

         }

         catch (Exception e)

         {
```

```
            CrestronConsole.PrintLine("Error UI set BasicTriList
touchPanel, uint join, bool value" + e.Message);
} }
        public static void set(uint join, string value)
        {
try {
            for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].StringInput[join].StringValue = value;
            }
          }
          catch (Exception e)
          {
            CrestronConsole.PrintLine("Error UI set BasicTriList
touchPanel, uint join, string value" + e.Message);
          }
        }
        public static void set(uint join, ushort value)
        {
```

```
try {
            for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].UShortInput[join].UShortValue = value;
            }
          }
          catch (Exception e)
          {
             CrestronConsole.PrintLine("Error UI set BasicTriList
touchPanel, uint join, ushort value" + e.Message);
} }
       public static void setSO(uint so, uint join, bool value)
       {
try {
                for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].SmartObjects[so].BooleanInput[join].Bool
Value = true;
```

```
} }
else {
                for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].SmartObjects[so].BooleanInput[join].Bool
Value = false;
} }
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Error UI setSO - " +
if (value) {
e.Message);
        }
}
    public static void setSO(uint so, uint join, string value)
    {
try {
            for (int t = 0; t <
```

```
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].SmartObjects[so].StringInput[join].Strin
gValue = value;
} }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Error UI setSO - " +
e.Message);
        }
    }
    public static void set_SO(uint so, uint join, bool value)
    {
try {
            for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].SmartObjects[so].BooleanInput[1].BoolVal
ue = value;
} }
```

```
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Error UI setSO - " +
e.Message);
        }
    }
    public static void setSO(uint so, int data)// ushort value
    {
        CrestronConsole.PrintLine("TEST 1");
        CrestronConsole.PrintLine("PANEl - " +
ControlSystem.allPanelList.Count().ToString());
        ControlSystem.allPanelList.Count();
+)
for (int t = 0; t < ControlSystem.allPanelList.Count(); t+
{
ControlSystem.allPanelList[t].SmartObjects[so].UShortInput["Set Num of
Items"].UShortValue = (ushort)(data); // magic string
            CrestronConsole.PrintLine("TEST 2");
        }
    }
```

```
public static void setLevel(uint join, ushort value)
{
try {
for (int t = 0; t <
ControlSystem.allPanelList.Count(); t++)
{
ControlSystem.allPanelList[t].UShortInput[join].UShortValue = value;
}
}
catch (Exception e)
{
e.Message);
}
}
} }
CrestronConsole.PrintLine("Error setLevel - " +
```

CONTROL SYSTEM

```
using System;
```

```
using Crestron.SimplSharp;
Basic SIMPL# Classes
using Crestron.SimplSharpPro;
Basic SIMPL#Pro classes
using Crestron.SimplSharpPro.CrestronThread;
Threading
using Crestron.SimplSharpPro.Diagnostics;
System Monitor Access
using Crestron.SimplSharpPro.DeviceSupport;
Generic Device Support
using Crestron.SimplSharp.CrestronIO;
using Crestron.SimplSharpPro.UI;
using UI_Base_Extensions;
using System.Collections.Generic;
//using System.Reflection;
using Crestron.SimplSharp.Reflection;
using Scheduler_MH;
namespace ScheduleApp

{

   public class ControlSystem : CrestronControlSystem

   {

   •       // For

   •       // For

   •       // For

   •       // For
```

- // For

private static CTimer internalTimer, internalTimer2,

internalTimer3;

//

bool push, SO, SchedulerDriver = false;

private Assembly SchedulerAssembly;

private Assembly SchedulerAssembly;

private Assembly keypadAssembly;

/// <summary>

/// ControlSystem Constructor. Starting point for the

SIMPL#Pro program.

/// Use the constructor to:

/// * Initialize the maximum number of threads (max = 400)

/// * Register devices

/// * Register event handlers

/// * Add Console Commands

///

/// Please be aware that the constructor needs to exit

quickly; if it doesn't

```
        /// exit in time, the SIMPL#Pro program will exit.
        ///
        /// You cannot send / receive data in the constructor
        /// </summary>
        ///
        private Scheduler SchedulerOffice;
        public static XpanelForSmartGraphics TP1;
        public static List<XpanelForSmartGraphics> allPanelList = new
List<XpanelForSmartGraphics>();
        public ControlSystem()
            : base()
{
try
            {
                Thread.MaxNumberOfUserThreads = 20;
                //Subscribe to the controller events (System, Program,
and Ethernet)
                CrestronEnvironment.SystemEventHandler += new
SystemEventHandler(ControlSystem_ControllerSystemEventHandler);
                CrestronEnvironment.ProgramStatusEventHandler += new
```

```
ProgramStatusEventHandler(ControlSystem_ControllerProgramEventHandler)
;
            CrestronEnvironment.EthernetEventHandler += new
EthernetEventHandler(ControlSystem_ControllerEthernetEventHandler);
        }
        catch (Exception e)
        {
e.Message);
        }
}
e.Message);
        }
}
ErrorLog.Error("Error in the constructor: {0}",
public override void InitializeSystem()
{
try {
      Initialize_XPanel();
   }
   catch (Exception e)
```

```
  {
internalTimer = new CTimer(CB, null, 2000, 1000);
ErrorLog.Error("Error in InitializeSystem: {0}",
      public void CB(object obj)
      {
try {
            SchedulerAssembly =
Assembly.LoadFrom("Scheduler_MH.dll");
            CrestronConsole.PrintLine("Assembly Full Name " +
SchedulerAssembly.FullName.ToString());
            CType myType =
SchedulerAssembly.GetType("Scheduler_MH.Scheduler");
            CType[] constructorTypes = new CType[]
{ typeof(string), typeof(int), typeof(int), typeof(int), typeof(int),
typeof(int),
typeof(int), typeof(int), typeof(int), typeof(int), typeof(int),
typeof(int)};
            ConstructorInfo constructorInfoObj =
myType.GetConstructor(BindingFlags.Instance | BindingFlags.Public,
null, constructorTypes, null);
```

```
            if (constructorInfoObj != null)
            {
                CrestronConsole.PrintLine("Constructor driver
properties - " + constructorInfoObj.ToString() + " - initializing
driver....");
                Object newScheduler =
constructorInfoObj.Invoke(new object[] { "Office", 5, 6, 7, 5, 9, 16,
6, 14, 63, 10, 78 });
                SchedulerOffice = (Scheduler)newScheduler;
                SchedulerOffice.s1_DeleteAnalogEvent += new
Scheduler.DeleteAnalog(SchedulerOfficeEvents.DeleteAnalogEvent);
                SchedulerOffice.s1_EditAnalogEvent += new
Scheduler.EditAnalog(SchedulerOfficeEvents.EditAnalogEvent);
                SchedulerOffice.s1_DeleteMaskSubPageEvent += new
Scheduler.DeleteMaskSubPage(SchedulerOfficeEvents.DeleteMaskSubPageEve
nt);
                SchedulerOffice.s1_EventNames += new
Action<List<string>>(SchedulerOfficeEvents.EventsList);
                SchedulerOffice.s1_ListMaskSubPageEvent += new
Scheduler.ListMaskSubPage(SchedulerOfficeEvents.ListMaskSubPageEvent);
```

```
            SchedulerOffice.s1_EditEvent += new
Scheduler.EditSchedule(SchedulerOfficeEvents.EditEvent);
            SchedulerOffice.s1_DeleteMaskSubPageEvent += new
Scheduler.DeleteMaskSubPage(SchedulerOfficeEvents.DeleteMaskSubPage);
            SchedulerOffice.Schedule_Data += new
Scheduler.Info_EditData(SchedulerOfficeEvents.Schedule_Data);
            SchedulerOffice.Task += new
Scheduler.ScheduleTask(SchedulerOfficeEvents.Task);
            SchedulerOffice.s1_SaveingEvent += new
Scheduler.SaveingSchedule(SchedulerOfficeEvents.SaveingEvent);
            //load UI on add subscribes
            SchedulerOffice.SchedulerSubscribesEvents += new
Action<bool>(SchedulerOfficeEvents.SubscribesEvents);
}
            internalTimer.Stop();
        }
        catch (RestrictionViolationException e)
        {
            CrestronConsole.PrintLine("Error CB - not allowed in
Crestron's sandbox due to 3 series sandbox restrictions ", e.Message);
```

```
        }
        catch (FileNotFoundException e)
        {
            CrestronConsole.PrintLine("Error CB - assembly Ref is
not found", e.Message);
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Error CB2 - ", e.Message);
            internalTimer2.Stop();
        }
finally {
            try{internalTimer2.Stop();}
catch { } }
}
    void Initialize_XPanel()
    {
try {
OnlineStatusChangeEventHandler(TP_OnlineStatusChange);
            allPanelList.Add(TP1);
```

```
            TP1.SigChange += new SigEventHandler(TP_SigChange);
//              string SGDPath = string.Format("{0}\
\ScheduleApp.sgd", Directory.GetApplicationDirectory());
            string SGDPath = string.Format("{0}\\Signals\
\ScheduleApp.sgd", Directory.GetApplicationDirectory());
            //string _driverFilename = string.Format("{0}\
\Extensions\\" + "schedule_3.json",
Directory.GetApplicationDirectory());
            if (TP1.Register() ==
eDeviceRegistrationUnRegistrationResponse.Success)
{
TP1 = new XpanelForSmartGraphics(0x16, this);
TP1.Description = "Xpanel";
TP1.OnlineStatusChange += new
                TP1.SigChange += new
SigEventHandler(TP_SigChange);
                TP1.LoadSmartObjects(SGDPath);
                foreach (KeyValuePair<uint, SmartObject> mySo in
TP1.SmartObjects)
                    mySo.Value.SigChange += new
```

```
SmartObjectSigChangeEventHandler(SmartObject_SigChange);
}
            UI.set(8, true);
        }
        catch (Exception e)
        {
            CrestronConsole.PrintLine("Error Initialize_XPanel ",
            ErrorLog.Notice("Error Initialize_XPanel ",
e.Message);
e.Message);
        }
}
    void TP_OnlineStatusChange(GenericBase currentDevice,
OnlineOfflineEventArgs args)
    {
    }
    void SmartObject_SigChange(GenericBase currentDevice,
SmartObjectEventArgs args)
    {
try {
```

```
                    CrestronConsole.PrintLine("SO ID " +
args.SmartObjectArgs.ID.ToString() + "  SO ITEM " +
args.Sig.UShortValue.ToString());
SchedulerOffice.Click(Convert.ToInt32(args.SmartObjectArgs.ID),
Convert.ToInt32(args.Sig.UShortValue));
} }
else {
RELEASE");
}
if (args.Sig.BoolValue)
{
   if (SO == false)
   {
SO = true;
SO = false;
CrestronConsole.PrintLine("SmartObject_SigChange -
}
 catch (Exception e)
 {
CrestronConsole.PrintLine("Value_SigChange Error " +
```

```
e.Message);

        }

}

      void TP_SigChange(BasicTriList currentDevice, SigEventArgs

args)

{
try

        {

            CrestronConsole.PrintLine("TP_SigChange " +

args.Sig.Number.ToString());

            switch (args.Sig.Type)

               {

                  case eSigType.Bool:

                     {

                        if (args.Sig.BoolValue)

                        {

                           if (push == false)

                           {

                              push = true;

                              if (SchedulerOffice != null)
```

```
SchedulerOffice.Click(Convert.ToInt32(args.Sig.Number));
                                else
CrestronConsole.PrintLine("Scheduler Driver is not enable");
                            }
e.Message);
} else {
                    push = false ;
                }
break; }
        case eSigType.UShort:
        case eSigType.String:
        case eSigType.NA:
        default:
break; }
}
catch (Exception e)
{
   CrestronConsole.PrintLine("TP_SigChange Error" +
   ErrorLog.Error("TP_SigChange Error" + e.Message);
}
```

```
}
/// <summary>
        /// Event Handler for Ethernet events: Link Up and Link Down.
        /// Use these events to close / re-open sockets, etc.
        /// </summary>
        /// <param name="ethernetEventArgs">This parameter holds the
values
        /// such as whether it's a Link Up or Link Down event. It will
also indicate
        /// wich Ethernet adapter this event belongs to.
        /// </param>
        void
ControlSystem_ControllerEthernetEventHandler(EthernetEventArgs
ethernetEventArgs)
        {
            switch (ethernetEventArgs.EthernetEventType)
            {//Determine the event type Link Up or Link Down
                case (eEthernetEventType.LinkDown):
                    //Next need to determine which adapter the event
is for.
```

```
external networks.

EthernetAdapterType.EthernetLANAdapter)

{
//

}

                break;

            case (eEthernetEventType.LinkUp):

                if (ethernetEventArgs.EthernetAdapter ==

EthernetAdapterType.EthernetLANAdapter)

{

}

break; }

        }

        /// <summary>

        /// Event Handler for Programmatic events: Stop, Pause,

Resume.

        /// Use this event to clean up when a program is stopping,

pausing, and resuming.

        /// This event only applies to this SIMPL#Pro program, it

doesn't receive events
```

```
        /// for other programs stopping
        /// </summary>
        /// <param name="programStatusEventType"></param>
        void
ControlSystem_ControllerProgramEventHandler(eProgramStatusEventType
programStatusEventType)
        {
            switch (programStatusEventType)
//LAN is the adapter is the port connected to
if (ethernetEventArgs.EthernetAdapter ==
            {
                case (eProgramStatusEventType.Paused):
                    //The program has been paused.  Pause all user
threads/timers as needed.
                    break;
                case (eProgramStatusEventType.Resumed):
                    //The program has been resumed. Resume all the
user threads/timers as needed.
                    break;
                case (eProgramStatusEventType.Stopping):
```

```
            //The program has been stopped.
            //Close all threads.
            //Shutdown all Client/Servers in the system.
            //General cleanup.
            //Unsubscribe to all System Monitor events
            break;
}
    }
    /// <summary>
    /// Event Handler for system events, Disk Inserted/Ejected,
and Reboot
    /// Use this event to clean up when someone types in reboot,
or when your SD /USB
    /// removable media is ejected / re-inserted.
    /// </summary>
    /// <param name="systemEventType"></param>
    void
ControlSystem_ControllerSystemEventHandler(eSystemEventType
systemEventType)
    {
```

```
        switch (systemEventType)
        {
            case (eSystemEventType.DiskInserted):
                //Removable media was detected on the system
                break;
            case (eSystemEventType.DiskRemoved):
                //Removable media was detached from the system
                break;
            case (eSystemEventType.Rebooting):
                //The system is rebooting.
                //Very limited time to preform clean up and save
any settings to disk.
                break;
}
}}
}
```

JSON_DATA

```
{
   "ListOfSchedules": [{
        "Name": "event_1",
        "Hr": 1,
        "Min": 22,
```

```
            "SelectTask":"System Off",
            "Sun": true,
            "Mon": true,
            "Tue": true,
            "Wed": false,
            "Thu": false,
            "Fri": false,
            "Sat": false
        }, {
            "Name": "event_2",
            "Hr": 24,
            "Min": 0,
            "SelectTask":"System Off",
            "Sun": true,
            "Mon": true,
            "Tue": false,
            "Wed": false,
            "Thu": false,
            "Fri": false,
            "Sat": false
},{

},{

"Name": "event_3",
"Hr": 12,
"Min": 55,
"SelectTask":"System Off",
"Sun": true,
"Mon": true,
"Tue": true,
"Wed": true,
"Thu": true,
"Fri": true,
"Sat": true
```

"Name": "event_4",
"Hr": 4,
"Min": 22,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,
"Tue": false,
"Wed": false,
"Thu": false,
"Fri": false,
"Sat": false
}, {

}, {

}, {

}, {

"Name": "event_5",
"Hr": 17,
"Min": 18,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,
"Tue": false,
"Wed": true,
"Thu": false,
"Fri": false,
"Sat": true
"Name": "event_6",
"Hr": 8,
"Min": 22,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,

```
"Tue": true,
"Wed": true,
"Thu": false,
"Fri": false,
"Sat": false
"Name": "event_7",
"Hr": 9,
"Min": 59,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,
"Tue": true,
"Wed": true,
"Thu": false,
"Fri": false,
"Sat": false
"Name": "event_8",
"Hr": 5,
"Min": 14,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,
"Tue": true,
"Wed": true,
"Thu": false,
"Fri": true,
} ]

}

   "Sat": true
},
{
   "Name": "event_9",
   "Hr": 23,
```

```
    "Min": 22,
    "SelectTask":"System Off",
    "Sun": true,
    "Mon": true,
    "Tue": true,
    "Wed": true,
    "Thu": true,
    "Fri": true,
    "Sat": true
},{

"Name": "event_10",
"Hr": 23,
"Min": 0,
"SelectTask":"System Off",
"Sun": true,
"Mon": false,
"Tue": false,
"Wed": false,
"Thu": false,
"Fri": false,
"Sat": false
```


VIDEO EXAMPLE OF THIS CODE

Cross platform & scheduler module - Write Once, Run Anywhere

https://www.linkedin.com/posts/michaelshanehaynes_crestron-amx-android-activity-6622153028046860289-4YIS