```
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Globalization;
using System.Linq;
using System.Reflection;
using System.Text;

namespace RaspiHomePiFaceDigital2
{
    public class ModelPiFaceDigital2
    {
        #region Fields
        #region Constants
        #endregion

        #region Variables
        private ViewPiFaceDigital2 _vPiFace;

        private List<Component> _components;
        private CommunicationWithServer _comWithServer;

        // Command to know
        private List<string> _raspiHomeComponentKnown = new List<string>()
        {
            "lumiere","lumieres",
            "store","stores",
            "television","televisions",
            "porte","portes",
            "fenetre","fenetres",
        };

        private List<string> _raspiHomeActionKnown = new List<string>()
        {
            "allumer","allume",
            "eteindre","eteins",
            "monter","monte",
            "descendre","descends",
            "stopper","stop",
            "ouvrir","ouvre",
            "fermer","ferme",
            "stopper","stop",
```

```csharp
        };

        // Word translation
        private Dictionary<string, string> _raspiLanguageTranslation = new Dictionary<string, string>()
        {
            { "lumiere","Light"}, { "lumieres","Light"},
            { "store","Store"}, { "stores","Store"},
        };

        // KEY=[ACTION NAME], VALUE[KEY=[PROPERTY NAME], VALUE=[VALUE TO SET THEPROPERTY]]
        private Dictionary<string, Dictionary<string, bool>> _raspiBooleanCommandTranslation = new Dictionary<string, Dictionary<string, bool>>()
        {
            { "allume", new Dictionary<string, bool> { { "IsOn", true } } }, { "allumer", new Dictionary<string, bool> { { "IsOn", true } } },
            { "eteins", new Dictionary<string, bool> { { "IsOn", false } } }, { "eteindre", new Dictionary<string, bool> { { "IsOn", false } } },
            { "monte", new Dictionary<string, bool> { { "IsUp", true } } }, { "monter", new Dictionary<string, bool> { { "IsUp", true } } },
            { "descends", new Dictionary<string, bool> { { "IsDown", true } } }, { "descendre", new Dictionary<string, bool> { { "IsDown", true } } },
            { "stop",new Dictionary<string, bool> { {"IsStop",true } } }, {"stopper",new Dictionary<string, bool> { {"IsStop",true } } },
        };
        #endregion
        #endregion

        #region Properties
        public ViewPiFaceDigital2 VPiFace
        {
            get
            {
                return _vPiFace;
            }

            set
            {
                _vPiFace = value;
            }
        }

        public List<Component> Components
        {
            get
            {
                return _components;
            }

            set
            {
                _components = value;
            }
        }

        public CommunicationWithServer ComWithServer
        {
            get
            {
                return _comWithServer;
            }

            set
            {
                _comWithServer = value;
            }
        }
        #endregion

        #region Constructors
        /// <summary>
        /// Constructor: Initializer
        /// </summary>
        /// <param name="paramView"></param>
        public ModelPiFaceDigital2(ViewPiFaceDigital2 paramView)
        {
            // Communication like Model-View
            this.VPiFace = paramView;

            // Initialize the components and add the components linked with
              the Raspberry
            this.Components = new List<Component>();
            this.Components.Add(new Light());
            this.Components.Add(new Store());

            // Initilize the PiFace Digital 2
            InitializePiFace();

            // Initialize the server communication
            this.ComWithServer = new CommunicationWithServer(this);
        }
        #endregion

        #region Methods
        /// <summary>
        /// Initialize the PiFace Digital 2
        /// </summary>
        private async void InitializePiFace()
        {
            try
            {
                await MCP23S17.InitilizeSPI();

                MCP23S17.InitializeMCP23S17();
                MCP23S17.SetPinMode(0x00FF); // 0x0000 = all outputs,
                  0xffff=all inputs, 0x00FF is PIFace Default
                MCP23S17.PullupMode(0x00FF); // 0x0000 = no pullups,
                    0xffff=all pullups, 0x00FF is PIFace Default
                MCP23S17.WriteWord(0x0000); // 0x0000 = no pullups, 0xffff=all
                  pullups, 0x00FF is PIFace Default
            }
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
            }
        }

        /// <summary>
        /// Set the value to be writed on the PiFace
        /// </summary>
        /// <param name="messageRead"> message read from the server </param>
        public void SetValue(string messageRead)
        {
            // Initialize the message value
            string sentence = this.RemoveDiacritics(messageRead);
            string action = this.GetActionFromSentence(sentence);
            string actionValue = this.ReadValueOfSelectedComponent(action);
            string component = this.GetComponentFromSentence(sentence);
            Type componentType = this.GetComponentType(component);

            foreach (Component itemType in this.Components)
            {
                if (itemType.GetType() == componentType)
                {
                    this.WriteValue(itemType, action, itemType.GetType
                    ().GetProperty(actionValue));
                }
            }
        }

        /// <summary>
        /// Find location exist
        /// </summary>
        /// <param name="sentence"> sentence order</param>
        /// <returns> return the action linked to the action word </returns>
        private string GetActionFromSentence(string sentence)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
                if (this._raspiHomeActionKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Get the componnent called
        /// </summary>
        /// <param name="sentence"> sentence order </param>
        /// <returns> return the component linked to the component word </returns>
        private string GetComponentFromSentence(string sentence)
        {
            string result = "";
            string[] words = sentence.ToLower().Split(' ');

            foreach (var word in words)
            {
                if (this._raspiHomeComponentKnown.Contains(word))
                {
                    result = word;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Find all client who have the object in the sentence
        /// </summary>
        /// <param name="componentName"></param>
        /// <returns>the object type</returns>
        private Type GetComponentType(string componentName)
        {
            Type result = null;
            Type[] types = typeof(Component).GetTypeInfo().Assembly.GetTypes();

            foreach (var typeOfComonent in types)
            {
                if (typeOfComonent.Name == this._raspiLanguageTranslation[componentName])
                {
                    result = typeOfComonent;
                    break;
                }
            }

            return result;
        }

        /// <summary>
        /// Read properties value of classes
        /// </summary>
        /// <param name="actionName"> name used to change the good property </param>
        /// <returns> return the name of the property to change the value </returns>
        private string ReadValueOfSelectedComponent(string actionName)
        {
            string result = "";

            foreach (var actionKeys in this._raspiBooleanCommandTranslation.Keys)
                if (actionKeys == actionName)
                {
                    // Find the Value of the dictionary trough the inner dictionary to get the first value
                    result = this._raspiBooleanCommandTranslation[actionName].First().Key;
                    break;
                }

            return result;
        }

        /// <summary>
        /// Search the val to change
        /// </summary>
        /// <param name="component"> the component to write value </param>
        /// <param name="action"> the action (ON/OFF) </param>
        /// <param name="typeVariable"> the property to change value </param>
        private void WriteValue(Component component, string action, PropertyInfo typeVariable)
        {
            switch (typeVariable.PropertyType.Name)
            {
                case "Boolean":
                    // Set the new value dynamicaly with value registered in an boolean dictionary
                    typeVariable.SetValue(component, this._raspiBooleanCommandTranslation[action][typeVariable.Name]);
                    break;
                case "Double":
                    break;
                case "Int16":
                case "Int32":
                case "Int64":
                    break;
            }
        }

        /// <summary>
        /// Stack Overflow solution to delete accents in strings
        /// http://stackoverflow.com/questions/249087/how-do-i-remove-diacritics-accents-from-a-string-in-net
        /// </summary>
        /// <param name="sentence"> sentence with diacritics to remove </param>
        /// <returns> same sentence without diacritics </returns>
        private string RemoveDiacritics(string sentence)
        {
            var normalizedString = sentence.Normalize(NormalizationForm.FormD);
            var stringBuilder = new StringBuilder();

            foreach (var c in normalizedString)
            {
                var unicodeCategory = CharUnicodeInfo.GetUnicodeCategory(c);
                if (unicodeCategory != UnicodeCategory.NonSpacingMark)
                {
                    stringBuilder.Append(c);
                }
            }

            return stringBuilder.ToString().Normalize
              (NormalizationForm.FormC);
        }
        #endregion
    }
}
```

```
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

using System;
using System.Collections.Generic;
using System.Linq;
using Windows.Networking;
using Windows.Networking.Sockets;
using Windows.Storage.Streams;

namespace RaspiHomePiFaceDigital2
{
    public class CommunicationWithServer
    {
        #region Fields
        #region Constants
        // Default information to connect on the server
        private const int PORT = 54565;
        //// Need to be changed fo each configuration
        private const string IPSERVER = "10.134.97.117";// "192.168.2.8";

        // String format for connection of the client
        private const string FORMATSTRING = "Connection:IPRasp={0};Location=
          {1};Component={2}";
        private const string COMMUNICATIONSEPARATOR = "@";

        // Important need to be changed if it's another room!
        private const string LOCATION = "Salon";
        private const string RPINAME = "PiFace_" + LOCATION;

        private const int MESSAGE_FULL_LENGHT = 512;
        #endregion

        #region Variables
        private ModelPiFaceDigital2 _mPiFace;

        // Connection's variable
        private StreamSocket _socket = new StreamSocket();
        private StreamSocketListener _listener = new StreamSocketListener();
        private List<StreamSocket> _connections = new List<StreamSocket>();
        private bool _isConnected = false;
        private bool _connecting = false;
        #endregion
        #endregion

        #region Properties
        public ModelPiFaceDigital2 MPiFace
        {
            get
            {
                return _mPiFace;
            }

            set
            {
                _mPiFace = value;
            }
        }

        public StreamSocket Socket
        {
            get
            {
                return _socket;
            }

            set
            {
                _socket = value;
            }
        }

        public StreamSocketListener Listener
        {
            get
            {
                return _listener;
            }

            set
            {
                _listener = value;
            }
        }

        public List<StreamSocket> Connections
        {
            get
            {
                return _connections;
            }

            set
            {
                _connections = value;
            }
        }

        public bool IsConnected
        {
            get
            {
                return _isConnected;
            }

            set
            {
                _isConnected = value;
            }
        }

        public bool Connecting
        {
            get
            {
                return _connecting;
            }

            set
            {
                _connecting = value;
            }
        }
        #endregion

        #region Constructors
        /// <summary>
        /// Constructor: Initializer
        /// </summary>
        /// <param name="paramModel"></param>
        public CommunicationWithServer(ModelPiFaceDigital2 paramModel)
        {
            this.MPiFace = paramModel;

            Connect();
        }
        #endregion

        #region Methods
        #region Methods
        /// <summary>
        /// Connect the raspberry to the server
        /// </summary>
        private async void Connect()
        {
            try
            {
                this.Connecting = true;
                // wait a confirmation from the server
                await this.Socket.ConnectAsync(new HostName(IPSERVER),
                  PORT.ToString());
                SendForInitialize();
                this.Connecting = false;
                this.IsConnected = true;

                WaitForData(this.Socket);
            }
```

```csharp
            catch (Exception)
            {
                this.Connecting = false;
                this.IsConnected = false;
            }
        }

        /// <summary>
        /// Listen the traffic on the port
        /// </summary>
        private async void Listen()
        {
            this.Listener.ConnectionReceived += listenerConnectionReceived;
            await this.Listener.BindServiceNameAsync(PORT.ToString());
        }

        void listenerConnectionReceived(StreamSocketListener sender,
          StreamSocketListenerConnectionReceivedEventArgs args)
        {
            this.Connections.Add(args.Socket);

            WaitForData(args.Socket);
        }

        /// <summary>
        /// Send the message in input to output
        /// </summary>
        /// <param name="socket"> actual stream </param>
        /// <param name="message"> message to send </param>
        private async void SendMessage(StreamSocket socket, string message)
        {
            DataWriter dataWriter = new DataWriter(socket.OutputStream);
            var len = dataWriter.MeasureString(message); // Gets the UTF-8
              string length.
            dataWriter.WriteInt32((int)len);
            dataWriter.WriteString(message);
            var ret = await dataWriter.StoreAsync();
            dataWriter.DetachStream();
        }

        /// <summary>
        /// Send to initialize the raspberry to the server
        /// </summary>
        private void SendForInitialize()
        {
            // Message send:
              "@NAME@Connection:IPRASP=x.x.x.x;Location=y;Component=z,z"
            SendMessage(this.Socket, string.Format(COMMUNICATIONSEPARATOR +
              RPINAME + COMMUNICATIONSEPARATOR + FORMATSTRING, GetHostName(),
              LOCATION, GetComponent()));
        }

        /// <summary>
        /// Wait data readed if exist
        /// </summary>
        /// <param name="socket"></param>
        private async void WaitForData(StreamSocket socket)
        {
            DataReader dataReader = new DataReader(socket.InputStream);
            dataReader.InputStreamOptions = InputStreamOptions.Partial;
            var msglenght = dataReader.UnconsumedBufferLength;
            uint stringBytes = msglenght;


            try
            {
                // Read modification in the stream
                stringBytes = await dataReader.LoadAsync(MESSAGE_FULL_LENGHT);

                // read message
                string msg = dataReader.ReadString(stringBytes);

                // Send in return if the value exist
                if (msg != "")
                {
                    this.MPiFace.SetValue(msg);
                }
            }
            catch (Exception e)
            {
                string output = e.Message;

                if (msglenght < 1)
                    return;
            }

            // Restart loop to wait data
            WaitForData(socket);
        }

        /// <summary>
        /// Get the ip of the raspberry
        /// </summary>
        /// <returns>return a string like 192.168.1.2</returns>
        private string GetHostName()
        {
            List<string> IpAddress = new List<string>();
            var Hosts = Windows.Networking.Connectivity.NetworkInformation.GetHostNames().ToList();
            foreach (var Host in Hosts)
            {
                string IP = Host.DisplayName;
                IpAddress.Add(IP);
            }
            return IpAddress.Last();
        }

        /// <summary>
        /// Get component in the list of components
        /// </summary>
        /// <returns> return a usable string for the connection on the
```

```
            server</returns>
        private string GetComponent()
        {
            string result = "";
            int cnt = 0;
            foreach (var component in this.MPiFace.Components)
            {
                // Get the name of the class
                result += component.ToString().Split('.').Last();
                cnt++;
                // Add the component separator for the string format
                if (cnt < this.MPiFace.Components.Count)
                    result += ",";

            }

            return result;
        }
    }
    #endregion
    #endregion
}


```

```
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

namespace RaspiHomePiFaceDigital2
{
    public abstract class Component{}
}

```

```
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

namespace RaspiHomePiFaceDigital2
{
    public class Light : Component
    {
        #region Fields
        #region Constant
        // PiFace output
        private const byte RELAIA = PiFaceDigital2.RelayA;
        private const byte RELAIB = PiFaceDigital2.RelayB;

        // PiFace State
        private const byte OFF = MCP23S17.Off;
        private const byte ON = MCP23S17.On;
        #endregion

        #region Variable
        private bool _isOn = false;
        private bool _isOnA = false;
        private bool _isOnB = false;
        #endregion
        #endregion

        #region Properties
        public bool IsOn
        {
            get
            {
                return _isOn;
            }

            set
            {
                _isOn = value;
                this.IsOnA = value;
                this.IsOnB = value;
            }
        }

        public bool IsOnA
        {
            get
            {
                return _isOnA;
            }

            set
            {
                _isOnA = value;
                if (value)
                {
                    // Turn ON the light
                    MCP23S17.WritePin(RELAIA, ON);
                }
                else
                {
                    // Turn OFF the light
                    MCP23S17.WritePin(RELAIA, OFF);
                }
            }
        }

        public bool IsOnB
        {
            get
            {
                return _isOnB;
            }

            set
            {
                _isOnB = value;
                if (value)
                {
                    // Turn ON the light
                    MCP23S17.WritePin(RELAIB, ON);
                }
                else
                {
                    // Turn OFF the light
                    MCP23S17.WritePin(RELAIB, OFF);
                }
            }
        }
        #endregion

        #region Constructor
        #endregion

        #region Methods
        #endregion
    }
}

```

```
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

using System;
using System.Threading.Tasks;
using Windows.UI.Xaml;

namespace RaspiHomePiFaceDigital2
{
    public class Store : Component
    {
        #region Fields
        #region Constant
        // PiFace output for motor
        private const byte UP = PiFaceDigital2.LED4;
        private const byte DOWN = PiFaceDigital2.LED3;

        // PiFace State
        private const byte OFF = MCP23S17.Off;
        private const byte ON = MCP23S17.On;

        // Max value for store (totaly open)
        private const int MAX_LEVEL = 200; // Time span total = 19seconds 
          (raspberry latency)
        // Min value for store (totaly close)
        private const int MIN_LEVEL = 0;

        // Tick for timer
        private const int TICKS = 10;
        private const int TICK_SECOND = 1;
        #endregion

        #region Variable
        private DispatcherTimer _dTimerUp = new DispatcherTimer();
        private DispatcherTimer _dTimerDown = new DispatcherTimer();

        private bool _isUp = false;
        private bool _isDown = false;
        private bool _isOpen = false;
        private bool _isClose = false;
        private bool _isStop = false;

        private int _counterStopped = 0;
        #endregion
        #endregion

        #region Properties
        public bool IsUp
        {
            get
            {
                return _isUp;
            }

            set
            {
                _isUp = value;

                // Maximum level
                if (value && this.CounterStopped < MAX_LEVEL)
                {
                    this.SetLevel("IsUp");
                }
            }
        }

        public bool IsDown
        {
            get
            {
                return _isDown;
            }

            set
            {
                _isDown = value;

                // Minimum level
                if (value && this.CounterStopped > MIN_LEVEL)
                {
                    this.SetLevel("IsDown");
                }
            }
        }

        public bool IsOpen
        {
            get
            {
                return _isOpen;
            }

            set
            {
                _isOpen = value;

                if (value)
                {
                    this.SetLevel("IsOpen");
                }
            }
        }

        public bool IsClose
        {
            get
            {
                return _isClose;
            }

            set
            {
                _isClose = value;

                if (value)
                {
                    this.SetLevel("IsClose");
                }
            }
        }

        public bool IsStop
        {
            get
            {
                return _isStop;
            }

            set
            {
                _isStop = value;

                // Stop everything
                if (value)
                {
                    this._dTimerUp.Stop();
                    this._dTimerDown.Stop();
                    SetLevel("IsStop");
                    this.IsStop = false;
                }
            }
        }

        public int CounterStopped
        {
            get
            {
                return _counterStopped;
            }

            set
            {
                _counterStopped = value;

                // Store manager
                if (value == MAX_LEVEL)
                {
                    this._dTimerUp.Stop();
                    SetLevel("IsStop");
                    _counterStopped = MAX_LEVEL;
                }
                else if (value == MIN_LEVEL)
                {
                    this._dTimerDown.Stop();
                    SetLevel("IsStop");
                    _counterStopped = MIN_LEVEL;
                }
            }
        }
        #endregion

        #region Constructor
        public Store()
        {
            this._dTimerUp.Interval = new TimeSpan(TICKS);
            this._dTimerUp.Tick += _dTimerUp_Tick;

            this._dTimerDown.Interval = new TimeSpan(TICKS);
            this._dTimerDown.Tick += _dTimerDown_Tick;
        }

        private void _dTimerUp_Tick(object sender, object e)
        {
            this.CounterStopped++;
        }

        private void _dTimerDown_Tick(object sender, object e)
        {
            this.CounterStopped--;
        }
        #endregion

        #region Methods
        /// <summary>
        /// Set the level
        /// </summary>
        /// <param name="propertyName"></param>
        private async void SetLevel(string propertyName)
        {
            switch (propertyName)
            {
                case "IsUp":
                    this.IsDown = false;

                    MCP23S17.WritePin(DOWN, OFF);
                    MCP23S17.WritePin(UP, ON);

                    this.SetLevelUp();
                    break;
                case "IsDown":
                    this.IsUp = false;

                    MCP23S17.WritePin(UP, OFF);
```

```
                        MCP23S17.WritePin(DOWN, ON);

                        this.SetLevelDown();
                        break;
                    case "IsOpen":
                        this.IsClose = false;

                        this.SetLevel("IsUp");
                        await Task.Delay(TimeSpan.FromSeconds(TICK_SECOND));
                        
                        this.SetLevel("IsStop");
                        break;
                    case "IsClose":
                        this.IsOpen = false;

                        this.SetLevel("IsDown");
                        await Task.Delay(TimeSpan.FromSeconds(TICK_SECOND));
                        this.SetLevel("IsStop");
                        break;
                    case "IsStop":
                        this.IsUp = false;
                        this.IsDown = false;
                        this.IsOpen = false;
                        this.IsClose = false;

                        MCP23S17.WritePin(UP, OFF);
                        MCP23S17.WritePin(DOWN, OFF);
                        break;
                }
            }

            /// <summary>
            /// Set upper the level of the store
            /// </summary>
            private void SetLevelUp()
            {
                this._dTimerDown.Stop();
                this._dTimerUp.Start();
            }

            /// <summary>
            /// Set downer the level of the store
            /// </summary>
            private void SetLevelDown()
            {
                this._dTimerUp.Stop();
                this._dTimerDown.Start();
            }
            #endregion
        }
    }
```

```xml
<Page
    x:Class="RaspiHomePiFaceDigital2.ViewPiFaceDigital2"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:RaspiHomePiFaceDigital2"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">
</Page>

```

```csharp
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

using Windows.UI.Xaml.Controls;

// Pour plus d'informations sur le modèle d'élément Page vierge, consultez la 
  page http://go.microsoft.com/fwlink/?LinkId=402352&clcid=0x409

namespace RaspiHomePiFaceDigital2
{
    /// <summary>
    /// Une page vide peut être utilisée seule ou constituer une page de 
      destination au sein d'un frame.
    /// </summary>
    public sealed partial class ViewPiFaceDigital2 : Page
    {
        #region Fields
        #region Constants
        #endregion

        #region Variables
        private ModelPiFaceDigital2 _mPiFace;
        #endregion
        #endregion

        #region Properties
        public ModelPiFaceDigital2 MPiFace
        {
            get
            {
                return _mPiFace;
            }

            set
            {
                _mPiFace = value;
            }
        }
        #endregion

        #region Constructors
        /// <summary>
        /// Construcor: Initializer
        /// </summary>
        public ViewPiFaceDigital2()
        {
            this.InitializeComponent();

            this.MPiFace = new ModelPiFaceDigital2(this);
        }
        #endregion

        #region Methods
        #endregion
    }
}

```

```csharp
/*---------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*---------------------------------------------------*/

namespace RaspiHomePiFaceDigital2
{
    public class PiFaceDigital2
    {
        // Output
        public const byte LED0 = 0x08;     // I/O Direction Register
        public const byte LED1 = 0x09;      // 1 = Input (default), 0 = Output
        public const byte LED2 = 0x0A;     // MCP23x17 Input Polarity Register
        public const byte LED3 = 0x0B;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)
        public const byte LED4 = 0x0C;     // MCP23x17 Interrupt on Change Pin Assignements
        public const byte LED5 = 0x0D;      // 1 = Input (default), 0 = Output
        public const byte LED6 = 0x0E;     // MCP23x17 Input Polarity Register
        public const byte LED7 = 0x0F;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)

        // Input
        public const byte IN0 = 0x00;     // I/O Direction Register
        public const byte IN1 = 0x01;      // 1 = Input (default), 0 = Output
        public const byte IN2 = 0x02;     // MCP23x17 Input Polarity Register
        public const byte IN3 = 0x03;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)
        public const byte IN4 = 0x04;     // MCP23x17 Interrupt on Change Pin Assignements
        public const byte IN5 = 0x05;      // 1 = Input (default), 0 = Output
        public const byte IN6 = 0x06;     // MCP23x17 Input Polarity Register
        public const byte IN7 = 0x07;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)

        // Switch / Button
        public const byte Sw0 = IN0;     // I/O Direction Register
        public const byte Sw1 = IN1;      // 1 = Input (default), 0 = Output
        public const byte Sw2 = IN2;     // MCP23x17 Input Polarity Register
        public const byte Sw3 = IN3;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)

        // Relay
        public const byte RelayA = LED1;     // MCP23x17 Input Polarity Register
        public const byte RelayB = LED0;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)
    }
}

```

```csharp
/*--------------------------------------------------*\
 * Author    : Salvi Cyril
 * Date      : 7th juny 2017
 * Diploma   : RaspiHome
 * Classroom : T.IS-E2B
 *
 * Description:
 *      RaspiHomePiFaceDigital2 is a program who use
 *   a PiFace Digital 2, it's an electronic card who
 *   can be use to plug electronic component. This
 *   program use the PiFace Digital 2 to activate
 *   light and store.
\*--------------------------------------------------*/

using System;
using System.Diagnostics;
using System.Threading.Tasks;
using Windows.Devices.Enumeration;
using Windows.Devices.Spi;

namespace RaspiHomePiFaceDigital2
{
    public class MCP23S17
    {

        private const byte IODIRA = 0x00;     // I/O Direction Register
        private const byte IODIRB = 0x01;      // 1 = Input (default), 0 = Output
        private const byte IPOLA = 0x02;     // MCP23x17 Input Polarity Register
        private const byte IPOLB = 0x03;     // 0 = Normal (default)(low reads as 0), 1 = Inverted (low reads as 1)
        private const byte GPINTENA = 0x04;     // MCP23x17 Interrupt on Change Pin Assignements
        private const byte GPINTENB = 0x05;     // 0 = No Interrupt on Change (default), 1 = Interrupt on Change
        private const byte DEFVALA = 0x06;    // MCP23x17 Default Compare Register for Interrupt on Change
        private const byte DEFVALB = 0x07;     // Opposite of what is here will trigger an interrupt (default = 0)
        private const byte INTCONA = 0x08;     // MCP23x17 Interrupt on Change Control Register
        private const byte INTCONB = 0x09;     // 1 = pin is compared to DEFVAL, 0 = pin is compared to previous state (default)
        private const byte IOCONA = 0x0A;     // MCP23x17 Configuration Register
        private const byte IOCONB = 0x0B;    //     Also Configuration Register
        private const byte GPPUA = 0x0C;     // MCP23x17 Weak Pull-Up Resistor Register
        private const byte GPPUB = 0x0D;     // INPUT ONLY: 0 = No Internal 100k Pull-Up (default) 1 = Internal 100k Pull-Up
        private const byte INTFA = 0x0E;     // MCP23x17 Interrupt Flag Register
        private const byte INTFB = 0x0F;    // READ ONLY: 1 = This Pin Triggered the Interrupt
        private const byte INTCAPA = 0x10;    // MCP23x17 Interrupt Captured Value for Port Register
        private const byte INTCAPB = 0x11;    // READ ONLY: State of the Pin at the Time the Interrupt Occurred
        private const byte GPIOA = 0x12;      // MCP23x17 GPIO Port Register
        private const byte GPIOB = 0x13;      // Value on the Port - Writing Sets Bits in the Output Latch
        private const byte OLATA = 0x14;      // MCP23x17 Output Latch Register
        private const byte OLATB = 0x15;      // 1 = Latch High, 0 = Latch Low (default) Reading Returns Latch State, Not Port Value!

        public const byte On = 1;
        public const byte Off = 0;
        public const byte Output = 0;
        public const byte Input = 1;

        private const byte Address = 0x00;   // offset address if hardware addressing is on and is 0 - 7 (A0 - A2)
        private const byte BaseAddW = 0x40;  // MCP23S17 Write base address
        private const byte BaseAddR = 0x41;  // MCP23S17 Read Base Address
        private const byte HAEN = 0x08;  // IOCON register for MCP23S17, x08 enables hardware address so sent address must match hardware pins A0-A2


        private static UInt16 PinMode = 0XFFFF;     // default Pinmode for the MXP23S17 set to inputs
        private static UInt16 PullUpMode = 0XFFFF;     // default pullups for the MXP23S17 set to weak pullup
        private static UInt16 InversionMode = 0X0000;     // default invert to normal
        private static UInt16 PinState = 0X0000;     // default pinstate to all 0's

        /*RaspBerry Pi2  Parameters*/
        private const string SPI_CONTROLLER_NAME = "SPI0";  /* For Raspberry Pi 2, use SPI0                             */
        private const Int32 SPI_CHIP_SELECT_LINE = 0;       /* Line 0 maps to physical pin number 24 on the Rpi2, line 1 to pin 26        */

        private static byte[] readBuffer3 = new byte[3]; /*this is defined to hold the output data*/
        private static byte[] readBuffer4 = new byte[4]; /*this is defined to hold the output data*/
        private static byte[] writeBuffer3 = new byte[3];//register, then 16 bit value
        private static byte[] writeBuffer4 = new byte[4];//register, then 16 bit value

        private static SpiDevice SpiGPIO;
        public static async Task InitilizeSPI()
        {
            try
            {
                var settings = new SpiConnectionSettings
```

```
                    (SPI_CHIP_SELECT_LINE);
                settings.ClockFrequency = 1000000;// 10000000;
                settings.Mode = SpiMode.Mode0; //Mode0,1,2,3;  MCP23S17 needs mode 0

                string spiAqs = SpiDevice.GetDeviceSelector(SPI_CONTROLLER_NAME);
                var deviceInfo = await DeviceInformation.FindAllAsync(spiAqs);
                SpiGPIO = await SpiDevice.FromIdAsync(deviceInfo[0].Id, settings);
            }

            /* If initialization fails, display the exception and stop running */
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
                //statusText.Text = "\nSPI Initialization Failed";
            }
        }

        public static void InitializeMCP23S17()
        {
            WriteRegister8(IOCONA, HAEN);                   // enable the hardware address incase there is more than one chip
            WriteRegister16(IODIRA, PinMode);                // Set the default or current pin mode

        }
        public static void WriteRegister8(byte register, byte value)
        {
            // Direct port manipulation speeds taking Slave Select LOW before SPI action
            writeBuffer3[0] = (BaseAddW | (Address << 1));
            writeBuffer3[1] = register;
            writeBuffer3[2] = value;
            try
            {
                SpiGPIO.Write(writeBuffer3);
            }

            /* If initialization fails, display the exception and stop running */
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
                //statusText.Text = "\nFailed to Wrie to DAC";
            }// Send the byte
        }
        public static void WriteRegister16(byte register, UInt16 value)
        {
            writeBuffer4[0] = (BaseAddW | (Address << 1));
            writeBuffer4[1] = register;
            writeBuffer4[2] = (byte)(value >> 8);
            writeBuffer4[3] = (byte)(value & 0XFF);
            try
            {
                SpiGPIO.Write(writeBuffer4);
            }

            /* If initialization fails, display the exception and stop running */
            catch (Exception ex)
            {
                Debug.WriteLine(ex.Message);
                //statusText.Text = "\nFailed to Wrie to DAC";
            }
        }

        // Set the pin mode a pin at a time or all 16 in one go
        // any value other then Input is taken as output
        public static void setPinMode(byte pin, byte mode)
        {
            if (pin > 15) return;               // only a 16bit port so do a bounds check, it cant be less than zero as this is a byte value
            if (mode == Input)
            {
                PinMode |= (UInt16)(1 << (pin));               // update the pinMode register with new direction
            }
            else
            {
                PinMode &= (UInt16)(~(1 << (pin)));            // update the pinMode register with new direction
            }
            WriteRegister16(IODIRA, PinMode);                // Call the generic word writer with start register and the mode cache
        }
        public static void SetPinMode(UInt16 mode)
        {
            WriteRegister16(IODIRA, mode);
            PinMode = mode;
        }

        // Set the pullup a pin at a time or all 16 in one go
        // any value other than On is taken as off
        public static void pullupMode(byte pin, byte mode)
        {
            if (pin > 15) return;
            if (mode == On)
            {
                PullUpMode |= (UInt16)(1 << (pin));
            }
            else
            {
                PullUpMode &= (UInt16)(~(1 << (pin)));
            }
            WriteRegister16(GPPUA, PullUpMode);
        }
        public static void PullupMode(UInt16 mode)
        {
            WriteRegister16(GPPUA, mode);
```

```
                PullUpMode = mode;
            }

            // Set the inversion a pin at a time or all 16 in one go
            public static void InvertMode(byte pin, byte mode)
            {
                if (pin > 15) return;
                if (mode == On)
                {
                    InversionMode |= (UInt16)(1 << (pin - 1));
                }
                else
                {
                    InversionMode &= (UInt16)(~(1 << (pin - 1)));
                }
                WriteRegister16(IPOLA, InversionMode);
            }
            public static void InvertMode(UInt16 mode)
            {
                WriteRegister16(IPOLA, mode);
                InversionMode = mode;
            }

            // WRITE FUNCTIONS - BY WORD AND BY PIN

            public static void WritePin(byte pin, byte value)
            {
                if (pin > 15) return;
                if (value > 1) return;
                if (value == 1)
                {
                    PinState |= (UInt16)(1 << pin);
                }
                else
                {
                    PinState &= (UInt16)(~(1 << pin));
                }
                WriteRegister16(GPIOA, PinState);
            }
            public static void WriteWord(UInt16 value)
            {
                WriteRegister16(GPIOA, value);
                PinState = value;
            }

            // READ FUNCTIONS - BY WORD, BYTE AND BY PIN
            public static UInt16 ReadRegister16()
            {
                writeBuffer4[0] = (BaseAddR | (Address << 1));
                writeBuffer4[1] = GPIOA;
                writeBuffer4[2] = 0;
                writeBuffer4[3] = 0;
                SpiGPIO.TransferFullDuplex(writeBuffer4, readBuffer4);
                return convertToInt(readBuffer4);                         //   
                  Return the constructed word, the format is 0x(register value)
            }
```

```
        public static byte ReadRegister8(byte register)
        {        // This function will read a single register, and return it
            writeBuffer3[0] = (BaseAddR | (Address << 1));  // Send the MCP23S17 opcode, chip address, and read bit
            writeBuffer3[1] = register;
            SpiGPIO.TransferFullDuplex(writeBuffer3, readBuffer3);
            return readBuffer4[2]; // convertToInt(readBuffer);                              // Return the constructed word, the format is 0x(register value)
        }
        public static UInt16 ReadPin(byte pin)
        {
            if (pin > 15) return 0x00;                  // If the pin value is not valid (1-16) return, do nothing and return
            UInt16 value = ReadRegister16();                        // Initialize a variable to hold the read values to be returned
            UInt16 pinmask = (UInt16)(1 << pin);                        // Initialize a variable to hold the read values to be returned
            return ((value & pinmask) > 0) ? On : Off;  // Call the word reading function, extract HIGH/LOW information from the requested pin
        }

        private static UInt16 convertToInt(byte[] data)
        {
            // byte[0] = command, byte[1] register, byte[2] = data high, byte[3] = data low
            UInt16 result = (UInt16)(data[2] & 0xFF);
            result <<= 8;
            result += data[3];
            return result;
        }
    }
}
```