SimpleDialog FrameWork

1. Introduction

A Simple Dialog Framework Is a Library which Simplifies the usage of Simple dialog Mode on the Pad. It can be used along with Delphi Driver, Dot net Driver.

The library is an independent library and does not automatically call the Simple dialog API's either from "StepOverSignatureDevice1.dll" or from Dotnet driver. It has two events called the "APIEvent" and "FrameWorkEvents". It is the responsibility of the user to make use of these Events to either call appropriate API's from driver when APIEvent is triggered or handle FrameWorkEvents when that is triggered.

The API Events can be Various types such as initialization, Update display, clear display, draw on to display etc as you know the simple dialog has 2 display buffers to accomplish its tasks called the Current buffer(which is seen on the Screen) and a shadow buffer to which images can be drawn and the display can be switched by calling update display. This switches current and shadow buffer .The API Event triggers an appropriate event to indicate if drawing has to be performed on shadow buffer or current buffer, if drawing on shadow buffer it will also indicate to update display, in addition it will also indicate if the buffer has to be cleared before drawing an image on to it.

The FrameWorkEvents are triggered as a result of an action performed by the user on the pad such as pressing a button, checking or unchecking a checkbox etc.

A detailed Explanation wil be given in the Events Section

2. Installation

The Library can be a DLL or Nuget Package, the Dll is called the "SimpleDialogFrameWork.dll" and can be Downloaded from here (it´s also part of the old DeviceAPI Setup) and the Nuget package can be Downloaded from here(Location TBD)

A template project can be Downloaded From here(Location TBD)

if the Delphi driver is used in a Visual studio c# framework project it would have included StepOverSignatureDevice1.dll in the Project, See how to do that here

Download SimpleDialogFrameWork.dll and include it in your project.

Requirement is .netFramework 4.5.1

3. Initialization

Assuming the following Initialization is done for StepOverSignatureDevice1.dll

```
SigDev.EnumeratePadsFirst(out padSetting);
SigDev.setPadSettings(padSetting, "");
SigDev.OnDevicePenEventHandler += onDevicePenEventHandler;
SigDev.OnSignFinished += SignFinishedevent;
SigDev.SetDriverLong(5, 10000); // signfinished after 200 millisec
```

The Simple Dialog Framework after Initialization

```
SDFrameWork Fw = new SDFrameWork(); // Instantiation of Simple Dialog Framework
Fw.APIEvent += SimpledialogPadApi; // Subscribe to API Events
Fw.FrameWorkEvents += SDialogFrameWorkEventsHandler; // Suscribe to FrameWorkEvents
Fw.Initialize();  // Call this after the Events have been subscribed to Else initialize event will not be handled
```

4. Simple Dialog Framework Events

4.1 APIEvent

The API Event for "StepOverSignatureDevice1.dll" is as below and will not change for a given version of the Library. if the user is using "StepOverSignatureDevice1.dll" then the below API Event Handler can directly be copied in to the code only modifying the object name."SigDev" with the one used by the user to Instantiate "StepOverSignatureDevice1.dll ". For dotnet driver all the API's starting with "SigDev" has to be replaced with Appropriate API

```
private void SimpledialogPadApi(object sender, SDFWUpdateOption e)
{
```

```
            if (e.APIType == TypeOfAPI.Initialize)
            {
                SigDev.SetSimpleDialogDisplayOption(0, 3); // for delphi API there is no special Initialise for simpledialog, just clearing the
            both buffers as i
            }
            else if (e.APIType == TypeOfAPI.Buffer)
            {


                /*
                * There are three types of buffer Operation in a SimpledialogPadApi and they have to be done in the Order listed below
                * 1. BufferOperation
                * 2. DrawToBuffer
                * 3. Bufferupdate
                * if this Order is Not followed it may result in Erroneous Display
                */
                //------------------- BufferClear ------------------------------------
                if (e.Bufferoperation == BufferOperation.ClearBoth)
                  SigDev.SetSimpleDialogDisplayOption(0, 3);
                else if (e.Bufferoperation == BufferOperation.ClearCurrent)
                  SigDev.SetSimpleDialogDisplayOption(0, 1);
                else if (e.Bufferoperation == BufferOperation.ClearShadow)
                  SigDev.SetSimpleDialogDisplayOption(0, 2);
                else if (e.Bufferoperation == BufferOperation.CopyCurrentToShadow)
                  SigDev.SetSimpleDialogDisplayOption(2, 0);
                //--------------------------------------------------------------------


                //------------------- DrawToBuffer ------------------------------------
                if (e.Bmp != null)
                  e.Bmp.Save("Update.png", ImageFormat.Png);

                if (e.DrawToBuffer == BufferDraw.ToCurrent)
                  SigDev.SetSimpleDialogResourceImage("", "Update.png", e.Location.X, e.Location.Y, 1);
                else if (e.DrawToBuffer == BufferDraw.ToShadow)
                  SigDev.SetSimpleDialogResourceImage("", "Update.png", e.Location.X, e.Location.Y, 0);



                if (e.Bmp != null)
                 File.Delete("Update.png");

                //--------------------------------------------------------------------


                //------------------- Bufferupdate ------------------------------------
                if (e.Bufferupdate == BufferUpdate.Update)
                  SigDev.SetSimpleDialogDisplayOption(1, 0);
                //--------------------------------------------------------------------

            }
            else if (e.APIType == TypeOfAPI.StartSignature)
            {


                //------------------------------------ Driver API Set Signature Rectangle and start Sign ----------------------------------------------
                SigDev.CreateSignatureRectangle(e.Location.X, e.Location.Y, e.size.Width, e.size.Height, 31);
                SigDev.startCapture("", false, false, false, false, ref padSetting);
                // ------------------------------------------------------------------------------------------------------------------------------------


            }


        }
```

As can be Seen by the APIEvent Handler called SimpledialogPadApi, there are three types of API's which are required to be called defined by the Enum **TypeOfAPI** called **Initialize, Buffer, StartSignature.**

Initialize event is triggered when *Fw.Initialize();* is called, this initializes many internal simpledialog framework operation and then triggers this event so user can call Simple dialog Initialization API from driver, in case of "StepOverSignatureDevice1.dll " calling clear both buffers also initializes the simple dialog mode. Dotnet driver has an initialize APi for simple dialog mode

Buffer event is probaly triggered by a component to update an image it was most likely preceded by a FrameworkEvent trigger, It has three sub modes called **BufferClear, DrawToBuffer and Bufferupdate** and they have to be executed in that order or it may result in Erroneous operation

StartSignature is triggered after calling *Fw.StartSignatureInBox(sdfwargs.SceneName, sdfwargs.ComponentName);* . This function calculates where the Signature image is to be drawn on the Screen based on Scene Location and the Signature box location on the Scene and then triggers the APIEvent where the user can use the event to call driver API to setSignature Rectangle size and location and then start Start Signature Capture

4.2 FrameWorkEvent

A frame work event is triggered as a result of user action on the Component Displayed on the Pad such as Depressing a button, checking /unchecking Checkbox, Radiobuttons etc. If the Dialog has only one scene then the Event can be handled as below

```
public void SDialogFrameWorkEventsHandler(object obj, SDFWargs sdfwargs)
{

        switch (sdfwargs.Component)
        {


                case SDFWComponentType.Button:
                                                /* Handle Button Events */  break;
                case SDFWComponentType.RadioButton:
                                                /* Handle RadioButton Events */  break;
                case SDFWComponentType.CheckBox:
                                                /* Handle CheckBox Events */  break;
                case SDFWComponentType.KeyBoard:
                                                /* Handle KeyBoard Events */  break;
                case SDFWComponentType.TextField:
                                                /* Handle TextField Events */  break;
                case SDFWComponentType.RichText:
                                                /* Handle RichText Events */  break;
                case SDFWComponentType.Box:
                                                /* Handle Box Events */  break;
                default: break;


        }


}
```

SDialogFrameWorkEventsHandler gets two parameters object and an Enum SDFWargs . The Object contains the Object of the Component that the has triggred the Event and the user can cast the object to Appropriate component and can access all the properties of the componet like below

```
SimpleDialogFWButton button = (SimpleDialogFWButton)obj
```

Note1 : SimpleDialogFWButton is a component and will be explained in the Component Section

Note2 : See Template Application to See how componets can be Handled in various ways

if more than one scene is defined in the Dialog the it becomes Easier to handle it like below

```
public void SDialogFrameWorkEventsHandler(object obj, SDFWargs sdfwargs)
{


        if (sdfwargs.Component == SDFWComponentType.KeyBoard)
          HandleKeyboard(obj, sdfwargs);


        if (sdfwargs.SceneName == "Scene1")
          HandleScene1(obj, sdfwargs);
        else if (sdfwargs.SceneName == "Scene2")
          HandleScene2(obj, sdfwargs);
        else if (sdfwargs.SceneName == "Scene3")
          HandleScene3(obj, sdfwargs);
```

```
        if (sdfwargs.Component == SDFWComponentType.NoComponent)
        {

        }


}


public void HandleScene1(object obj, SDFWargs sdfwargs)
{

        switch (sdfwargs.Component)
        {


                case SDFWComponentType.Button:
                                        /* Handle Button Events */  break;
                case SDFWComponentType.RadioButton:
                                       /* Handle RadioButton Events */  break;
                case SDFWComponentType.CheckBox:
                                        /* Handle CheckBox Events */  break;
                case SDFWComponentType.KeyBoard:
                                        /* Handle KeyBoard Events */  break;
                case SDFWComponentType.TextField:
                                        /* Handle TextField Events */  break;
                case SDFWComponentType.RichText:
                                      /* Handle RichText Events */  break;
                case SDFWComponentType.Box:
                                     /* Handle Box Events */  break;
                default: break;


        }

}


and SO on
```

5. Creating Scene(dialog)

In order to create a dialog a base is required on which a components can be placed, this base is called a Scene. All the Components listed below can be placed on the Scene. A user can define As many Scenes As he/she wants.

A scene can be any size however it is not recommended for the size of Scene to be bigger than the pad-display size although it can be smaller , if the Scene is bigger that pad-display size then only part of the Scene is displayed and user will not have access to components drawn outside the pad-display. The Scene can be placed anywhere on the pad-display, provided user takes care not to display partial Scene else the user may have the same problem as described when the Scene size is bigger than the pad-display size

A scene can be created in two ways using an Image as a background in which case the Scene size is the Size of the Image. or user can define size and Background color. Two Simple Dialog Framework(SDFW) API's are used to Accomplish this task. The Return type is Error which is Defined at the Error Section

1. public Error AddScene(string name, Color BGMColor, Point Location, Size size)

SlNo	Arguments	Description	Remarks
1	name	This Parameter takes string as input.	This is a unique identifier if another Scene Exists with the Same name then this Scene cannot be created and Returns Error.SceneAlreadyExists
2	BGMColor	This parameter takes Color as input	The scene is filled with this color
3	Location	This Parameter takes Point as input	This can be used to display a scene anywhere on the pad-display with new Point(0,0) being at the Top left corner

4	size	This Parameter takes Size as input	This defines the size of the Scene
Eg :- Fw.AddScene("MainScene", Color .White, new Point (0,0), new Size (1024, 600));			

2. public Error AddScene(string name, string BGMImageFile, Point Location)

SlNo	Arguments	Description	Remarks
1	name	This Parameter takes string as input.	This is a unique identifier if another Scene Exists with the Same name then this Scene cannot be created and Returns Error.SceneAlreadyExists
2	BGMImageFile	This parameter takes string as input	This stores the location where the File is stored
3	Location	This Parameter takes Point as input	This can be used to display a scene anywhere on the pad-display with new Point(0,0) being at the Top left corner
Eg:- Fw.AddScene("DocumentViewScene", projectPath + @"\g13\SDFWTest\doc_p1.png", new Point (0, 0));			

6. KeyBoard

This Function Adds A User Defined Keyboard to the framework

public Error AddKeyBoardSDialogFrameWork(string Name, Size size, string RowColSize, string KeyBoardChars, Color? KeyBoardBgClr = null, Color? KeyBoardBtnClr = null, Color? KeyBoardBtnPrsdClr = null, string FontName = "Arial", int FontSize = 17, FontStyle FontStyle = FontStyle.Regular, Color? FontColor = null);

SlNo	Arguments	Description	Remarks
1	Name	This Parameter takes string as input.	Each Keyboard Is Identified by a name, The user has to give a uinque Name to each keyboard. User can define as many keyboards as is required and there are no Restriction on the no of Keyboards. However only one Keyboard can be Active (be displayed on the pad) at a time
2	size	This Parameter takes Size as input	This Parameter Defines the Size of the keyboard
3	RowColSize	This Parameter takes string as input.	This Parameter takes string as input. However this defines the Size and no of rows and coloumns of the Keyboard, the Rows are seperated by the character "|" and Columns are seperated by the character ",". Eg "*10=100| *9=111| *9=111| *2=100, 600, *2=100", here the keyboard has four rows seperated by three "|" character. The Coloumns defines the X size of each buttons and Y size is calculated as Y size = keyboard.Size.height/ No of rows It is possible to Define the Colomn width of individual buttons as values seperated comma like this "100, 100, 100, 100, 100, 100, 100, 100, 100", It is also possible if there are multiple buttons of same width in a particular row to be defined as "*10=100", in that case it has to be defined in the same order as "*NoofCols=WidthofEachButton", the First Identifier char * identifies the no of cols which is equal to the number following it, the second identifier is the char "=" which identifies the width of the buttons which again is equal to the number following it. So here the keyboards First row has ten buttons with a width of 100 pixels. It is also possible to to have a combination of repeated size definition and indivdual size defintion as defined in the last row of the example "*2=100, 600, *2=100". Here The last row has two buttons of size 100 pixels and one long button of 600 pixels and followed by another two buttons of 100pixels
4	KeyBoardChars	This Parameter takes string as input.	This Parameter takes string as input, As in the case of parameter "RowColSize" this also uses character "|", and "," to Split rows and coloumns respectively. However the repeated charcter like repated width is not possible as the each buttons are expected to have unque individual Charcters. The String "BS" is used for Backspace this is replaced by a back arrow while drawing the keyboard image, howver when the button is depressed then it return the string "BS" in the SimpleDialogFrameworkEvent if subscribed to it
5	KeyBoardBgClr	This parameter takes Color as input	Default color "Color.LightGray" It used as a backbround color of the Keybaord
6	KeyBoardBtnClr	This parameter takes Color as input	Default color "Color.Gray" It used as a button color of the Keybaordwhen button is not depressed
7	KeyBoardBtnPrsdClr	This parameter takes Color as input	Default color "Color.DarkGray" It used as a button color of the Keybaord when button is depressed
10	FontName	This parameter takes string as input	Default value is "Arial" The Name of the Font such as "Arial", "Times New Roman" etc

11	FontSize	This parameter takes int as input	Default value is 17 this defines the size of the text such as 12, ... 40..
12	Style	This parameter takes FontStyle as input	Default value is FontStyle.Regular this defines the FontStyle such as Regular, Bold, Italic
13	FontColor	This parameter takes Color as input	Default value is Color.White this defines the color of the font
Eg: Fw.AddKeyboard("SmallAlphabetsKeyBoard", new Size(1000, 200), " *10=100| *9=111| *9=111| *2=100, 600, *2=100", "q,w,e,r,t,y,u,i,o,p|a,s,d,f,g,h,j,k,l|ABC, z,x,c,v,b,n,m,BS|?123, , space, , Enter", Color.White, Color.Black, Color.Gray, "Arial", 24, FontStyle.Regular, Color.White);			

7. Adding components

The following are the List of Components

7.1 Button :

A button is a component which can be added to a Scene in two ways using Images such as Normal Image, Pressed image and disabled image in which case the button size is the Size of the Image. or user can define size and button color, pressed color and disabled color

Two Simple Dialog Framework(SDFW) API's are used to Accomplish this task. The Return type is Error which is Defined at the Error Section

public Error AddButton(string SceneName, string ButtonName, bool Enabled, Point Location, string buttonImageFile, string ButtonPressedImage, string ButtonDisabledImage, string ButtonText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

SlNo	Arguments	Description	Remarks
1	SceneName	This Parameter takes string as input.	This is a name of the Scene to which the button is to be added, if scene does not exist then it return error
2	ButtonName	This parameter takes string as input	This is a unique identifier if another button Exists with the Same name with in this scene then this button cannot be created and Returns Error.ComponentNameAlreadyExists. However same name for a button in different scene is allowed
3	Enabled	This Parameter takes bool as input	this Defines if the button is Enabled or disabled and loads appropriate image also if disabled then no event is triggered
4	Location	This Parameter takes Point as input	this defines where on the Scene the Button is to be placed
5	buttonImageFile	This parameter takes string as input	This stores the location where the File is stored
6	ButtonPressedImage	This parameter takes string as input	This stores the location where the File is stored
7	ButtonDisabledImage	This parameter takes string as input	This stores the location where the File is stored
8	ButtonText	This is an Optional parameter. This parameter takes string as input	Default value is null Text is Dispalyed if user has defined a text
9	FontName	This is an Optional parameter. This parameter takes string as input	If Text is null this parameter is unused Default value is "Arial" The Name of the Font such as "Arial", "Times New Roman" etc
10	FontSize	This is an Optional parameter. This parameter takes int as input	If Text is null this parameter is unused Default value is 12 this defines the size of the text such as 12, ... 40..
11	Style	This is an Optional parameter. This parameter takes FontStyle as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the FontStyle such as Regular, Bold, Italic

12	FontColor	This is an Optional parameter. This parameter takes Color as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the color of the font

public Error AddButton(string SceneName, string ButtonName, bool Enabled, Point ButtonLoc, Size ButtonSize, Color? ButtonColor = null, Color? ButtonPressedColor = null, Color? ButtonDisabledColor = null, string ButtonText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

SlNo	Arguments	Description	Remarks
1	SceneName	This Parameter takes string as input.	This is a name of the Scene to which the button is to be added, if scene does not exist then it return error
2	ButtonName	This parameter takes string as input	This is a unique identifier if another button Exists with the Same name with in this scene then this button cannot be created and Returns Error.ComponentNameAlreadyExists. However same name for a button in different scene is allowed
3	Enabled	This Parameter takes bool as input	this Defines if the button is Enabled or disabled and loads appropriate image also if disabled then no event is triggered
4	Location	This Parameter takes Point as input	this defines where on the Scene the Button is to be placed
5	ButtonSize	This Parameter takes Size as input	This defines the size of the Button
6	ButtonColor	This parameter takes Color as input	The scene is filled with this color
7	ButtonPressedColor	This parameter takes Color as input	The scene is filled with this color
8	ButtonDisabledColor	This parameter takes Color as input	The scene is filled with this color
9	ButtonText	This is an Optional parameter. This parameter takes string as input	Default value is null Text is Dispalyed if user has defined a text
10	FontName	This is an Optional parameter. This parameter takes string as input	If Text is null this parameter is unused Default value is "Arial" The Name of the Font such as "Arial", "Times New Roman" etc
11	FontSize	This is an Optional parameter. This parameter takes int as input	If Text is null this parameter is unused Default value is 12 this defines the size of the text such as 12, ... 40..
12	Style	This is an Optional parameter. This parameter takes FontStyle as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the FontStyle such as Regular, Bold, Italic
13	FontColor	This is an Optional parameter. This parameter takes Color as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the color of the font

7.2 RadioButton

A Radio button is a type of component where only one button is checked at any one point in time in a group, user can have as many groups of radio button as he/she wishes.

Similar to Button Radio Button can be created in two ways using Two Simple Dialog Framework(SDFW) API's . The Return type is Error which is Defined at the Error Section

public Error AddRadioButton(string SceneName, string RadioButtonName, int RadioButtonGroupNo, bool Enabled, Point RadioButtonLoc, bool CheckedState, string RadioButtonImageFile, string RadioButtonCheckedImageFile, string RadioButtonDisabledImage, string RadioButtonText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

SlNo	Arguments	Description	Remarks
1	SceneName	This Parameter takes string as input.	This is a name of the Scene to which the button is to be added, if scene does not exist then it return error
2	RadioButtonName	This parameter takes string as input	This is a unique identifier if another RadioButtonNameExists with the Same name with in this scene then this RadioButtonNamecannot be created and Returns Error.ComponentNameAlreadyExists. However same name for a RadioButtonName in different scene is allowed
3	RadioButtonGroupNo	This Parameter takes int as input	this defines to which group the radio button group belongs to user can add as many radiobuttons to a group as he/she wishes
4	Enabled	This Parameter takes bool as input	this Defines if the radiobutton is Enabled or disabled and loads appropriate image also if disabled then no event is triggered
5	RadioButtonLoc	This Parameter takes Point as input	this defines where on the Scene the RadioButton is to be placed
6	CheckedState	This Parameter takes bool as input	this Defines if the radiobutton is Checked or Not and loads appropriate image
7	RadioButtonImageFile	This parameter takes string as input	This stores the location where the File is stored
8	RadioButtonCheckedImageFile	This parameter takes string as input	This stores the location where the File is stored
9	RadioButtonDisabledImage	This parameter takes string as input	This stores the location where the File is stored
10	RadioButtonText	This is an Optional parameter. This parameter takes string as input	Default value is null Text is Dispalyed if user has defined a text
11	FontName	This is an Optional parameter. This parameter takes string as input	If Text is null this parameter is unused Default value is "Arial" The Name of the Font such as "Arial", "Times New Roman" etc
12	FontSize	This is an Optional parameter. This parameter takes int as input	If Text is null this parameter is unused Default value is 12 this defines the size of the text such as 12, ... 40..
13	Style	This is an Optional parameter. This parameter takes FontStyle as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the FontStyle such as Regular, Bold, Italic
14	FontColor	This is an Optional parameter. This parameter takes Color as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the color of the font

public Error AddRadioButton(string SceneName, string RadioButtonName, int RadioButtonGroupNo, bool Enabled, Point RadioButtonLoc, bool CheckedState, int RadioButtonDiameter, Color? RadioButtonColor = null, Color? RadioButtonCheckedColor = null, Color? RadioButtonDisabledColor = null, string RadioButtonText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

SlNo	Arguments	Description	Remarks
1	SceneName	This Parameter takes string as input.	This is a name of the Scene to which the button is to be added, if scene does not exist then it return error
2	RadioButtonName	This parameter takes string as input	This is a unique identifier if another RadioButtonNameExists with the Same name with in this scene then this RadioButtonNamecannot be created and Returns Error.ComponentNameAlreadyExists. However same name for a RadioButtonName in different scene is allowed
3	RadioButtonGroupNo	This Parameter takes int as input	this defines to which group the radio button group belongs to user can add as many radiobuttons to a group as he/she wishes
4	Enabled	This Parameter takes bool as input	this Defines if the radiobutton is Enabled or disabled and loads appropriate image also if disabled then no event is triggered
5	RadioButtonLoc	This Parameter takes Point as input	this defines where on the Scene the RadioButton is to be placed

6	CheckedState	This Parameter takes bool as input	this Defines if the radiobutton is Checked or Not and loads appropriate image
7			
5	RadioButtonImageFile	This parameter takes string as input	This stores the location where the File is stored
6	RadioButtonCheckedImageFile	This parameter takes string as input	This stores the location where the File is stored
7	RadioButtonDisabledImage	This parameter takes string as input	This stores the location where the File is stored
8	RadioButtonText	This is an Optional parameter. This parameter takes string as input	Default value is null Text is Dispalyed if user has defined a text
9	FontName	This is an Optional parameter. This parameter takes string as input	If Text is null this parameter is unused Default value is "Arial" The Name of the Font such as "Arial", "Times New Roman" etc
10	FontSize	This is an Optional parameter. This parameter takes int as input	If Text is null this parameter is unused Default value is 12 this defines the size of the text such as 12, ... 40..
11	Style	This is an Optional parameter. This parameter takes FontStyle as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the FontStyle such as Regular, Bold, Italic
12	FontColor	This is an Optional parameter. This parameter takes Color as input	If Text is null this parameter is unused Default value is FontStyle.Regular this defines the color of the font

7.3 CheckBox

public Error AddCheckBox(string SceneName, string CheckBoxName, bool Enabled, Point CheckboxLoc, bool CheckedState, string CheckBoxImageFile, string CheckBoxCheckedImageFile, string CheckBoxDisabledImage, string CheckBoxText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

public Error AddCheckBox(string SceneName, string CheckBoxName, bool Enabled, Point CheckboxLoc, bool CheckedState, Size CheckBoxSize, Color? CheckBoxColor = null, Color? CheckBoxBgColor = null, Color? CheckBoxDisabledColor = null, string CheckBoxText = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

The Parameters explanation is Similar to Radio button

7.4 TextField

This creates a TextField to the Scene to which input can be entered via virtual Keyboard

public Error AddTextField(string SceneName, string TextFieldName, bool Enabled, Point TextFieldLoc, Size size, int MaxNoOfChars = 0, Color? BoxGroundColor = null, Color? BackGroundColor = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

The Parameters explanation is Similar to Radio button, button or checkbox

7.5 Label

This creates a label to the scene

public Error AddLabel(string SceneName, string LabelName, Point Location, string Text, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle.Regular, Color? FontColor = null)

The Parameters explanation is Similar to Radio button, button or checkbox

7.6 Image

This adds a image to the Scene

```
public  Error AddImage(string SceneName, string ImageName, Point Location, string ImageFile)
```

The Parameters explanation is Similar to Radio button, button or checkbox

7.7 Rectangle

This adds a Rectangle to the Scene

```
public  Error AddRectangle(string SceneName, string RectangleName, Point Location, Size size, Colorcolor)
```

The Parameters explanation is Similar to Radio button, button or checkbox

7.8 RichText

This creates a multi line TextField to the Scene to which input can be entered via virtual Keyboard

```
public Error AddRichText(string SceneName, string TextFieldName, bool Enabled, Point TextFieldLoc, Size size, int MaxNoOfChars = 0, Colo
? BoxGroundColor = null, Color? BackGroundColor = null, string FontName = "Arial", int FontSize = 12, FontStyle Style = FontStyle .Regular,
Color? FontColor = null)
```

The Parameters explanation is Similar to Radio button, button or checkbox

7.9 Box

This is a Box which creates a rectangle in which it is possible to capture signature or any hand writing. using the API *Fw.StartSignatureInBox (sdfwargs.SceneName, sdfwargs.ComponentName)*

if a Scenename and a box name is given then it will trigger an APIEvent for start signature with parameters which gives the absolute value of the location of the Box irrespective of where the Scene is placed on the Pad display.

```
 public Error  AddBox(string SceneName, string BoxName, bool Enabled, Point BoxLoc, Size size, Color? BoxColor = null, Color?
BackGroundColor = null, Color? DisabledColor = null)
```

The Parameters explanation is Similar to Radio button, button or checkbox except for the BoxColor which defines the Color of the box if no color is desired then enter Color.Transparent

8. Delete Components

8.1 Delete Indivudual components

Each individual components can be deleted using the API

```
public Error   DeleteElement(SDFWComponentType CompType, string ElementName, string ComponentName)
```

8.2 Delete Scene

```
public Error   DeleteScene(string SceneName)
```

9. Clear Components

9.1 Clear components with in scene

```
public Error ClearComponentList(string SceneName, SDFWComponentType type)
```

SceneName and Componenet type which has to be Cleared

9.2 Clear All Scenes

```
public Error ClearAllScenes()
```

9.2 ClearAllKeyBoards

```
public Error ClearAllKeyBoards()
```

10. Error

```
public enum Error : int
{
/// <summary>
/// Represents a success.
/// </summary>
SUCCESS = 0,
WRITE_OUTSIDE_RAM = 1,
READ_OUTSIDE_RAM = 2,
WRONG_SEKTOR = 3,
FLASH_FAILURE = 4,
FUNCTION_NOT_IMPLEMENTED = 5,
WRONG_HASH = 6,
IN_PROGRESS = 7,
INVALID_SECTOR = 8,
SECTOR_READ_PROTECTED = 9,
SECTOR_WRITE_PROTECTED = 10,
SECTOR_ERASE_PROTECTED = 11,
WRITE_OUTSIDE_SECTOR = 12,
READ_OUTSIDE_SECTOR = 13,
ADMINKEY_SET = 14,
ADMINKEY_NOT_SET = 15,
ADMINKEY_ALREADY_SET = 16,
WRONG_ADMINKEY = 17,
PAD_IS_LOCKED = 18,
PRE_DOCHASH_NOT_SET = 19,
FINAL_DOCHASH_NOT_SET = 20,
AESCONTEXT_NOT_SET = 21,
THERE_IS_NO_SIGNBMP = 22,
NOT_IN_SIGNATURE_MODE = 23,
RSA_ENCRYPTION_FAILED = 24,
RSA_CHECK_PUBKEY_FAILED = 25,
PADMODE_IS_ALREADY_SET = 26,
SWITCHMODE_NOT_ALLOWED = 27,
INVALID_ADDRESS = 28,
FILE_ALREADY_OPEN = 29,
FOPEN_FAILED = 30,
FILE_NOT_OPEN = 31,
FCLOSE_FAILED = 32,
DELETE_FAILED = 33,
FILE_READ_FAILED = 34,
NO_DIRECTORY_ENTRY = 35,
DIFFERENT_LENGTH_WRITTEN = 36,
OUT_OF_MEMORY = 37,
FORMAT_FAILED = 38,
NOT_IN_CONFIGURATION_MODE = 39,
ACTION_NOT_ALLOWED = 40,
INVALID_VALUE = 41,
COMMAND_NOT_IMPLEMENTED = 42,
GENERAL_FAILURE = 43,
HID_STREAM_FAILURE = 45,
UNKNOWN_ERROR = 0xFF,
```

```
// com errors are from 0x00 to 0xff
ExceptionCaught = 0x100,
ArgumentOutOfRange,
DeviceNotConnected,
CertificateNotFound,
RsaDecryptionFailed,
AesDecryptionFailed,
NotInSimpleDialogMode,
FileNotFound,
FileUploadFailed,
IncorrectModeValue,
NotInValidMode,
InvalidFileFormat,
BackGroundImageNotSet,
WriteBufFail,
InvalidSignRecttParameters,
FileCloseFailed,
UserHasCancelledOperation,
InvalidLicense,
ContinueWithoutLicense,
ReadingSignatureImageFailed,
TransferFailed,
FileNotFoundOnDevice,
ConnectionLost,
NotNetFrameWork,
TabletPcSignFormAlreadyDefined,
SceneNotInitialized,
ComponentNotInitialized,
SceneAlreadyExists,
ComponentNameAlreadyExists,
ComponentDoesNotExists,
SimpleDialogFrameWorkNotInitialized,
InvalidRadioButtonGroup,
InvalidSceneName,
InvalidComponentName,
NoSignBoxIsActive,
UnknownPad, // keep this value at the end //why?
}
```