# Integration Guide

## pDoc SigEmbed SDK

**Version 1.5**
August 1, 2017

## Table of Contents

## Table of Contents

## Table of Contents

## Table of Contents

## Table of Contents

## Table of Contents

## Table of Contents

# Table of Contents

## 1.0 – Introduction

The Topaz pDoc SigEmbed SDK Module provides the capabilities to create standard DigSig fields in PDF documents, embed signatures in the DigSig fields, and verify signed DigSig fields, create form fields and form field filling. Designed for using in Windows environments, the pDoc SigEmbed SDK Module works in tandem with signature capture software, such as the Topaz iDoc SigCapture SDK, to protect the raw signature data and restrict the sources for such data. The Adobe “DigSig” digital signature standard is supported for standalone verification using Adobe Acrobat software.

The information shared within this pDoc SigEmbed SDK 2.0 Integration Document is written for a technical audience who will be integrating Topaz's pDoc SigEmbed SDK 2.0 into their applications. The document explains the various API calls that are exposed through this Platform and also provide some samples on how to use them. All the syntax as well as the samples provided in the document is in C# language.

## 2.0 – Overview

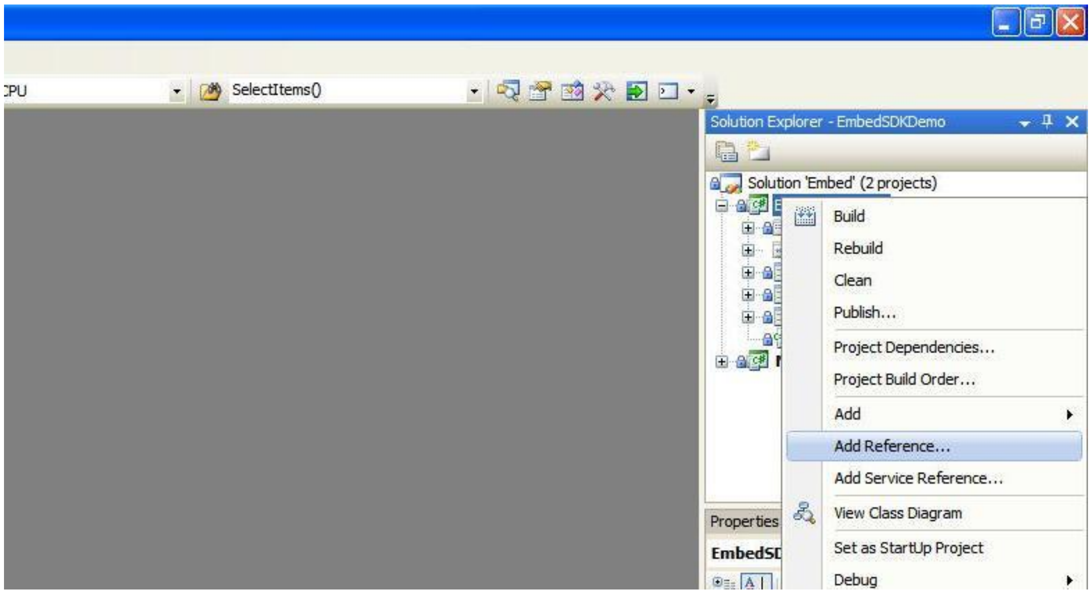
Topaz pDoc SigEmbed SDK 2.0 supports the following features.

- Processing of PDF documents protected against unauthorized access using Open Password.
- PDF Form field creation & deletion (TextBox, ComboBox, ListBox, CheckBox, RadioButton)
- PDF Form field styling
- PDF Form field filling (Embed Data)
- Creation & deletion of PDF signature fields
- Ability to add custom dictionaries to signature field
- Embedding signatures in PDF signature field
- Local (single step) and remote hash signing (two step signing)
- Verification of signed PDF signature field

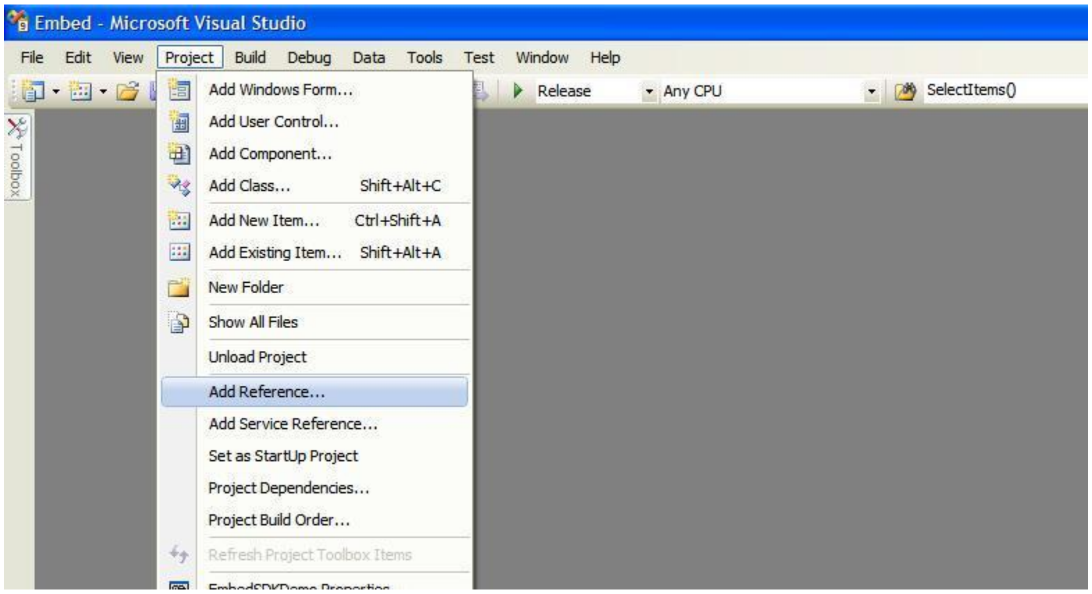
## 3.0 – Add pDoc SigEmbed SDK Reference To Your Project

Before you begin to use the various API calls described in this document you should add a reference to the Topaz's pDoc SigEmbed SDK 2.0 assembly to your project. This section describes how this can be done using Visual Studio 2008.

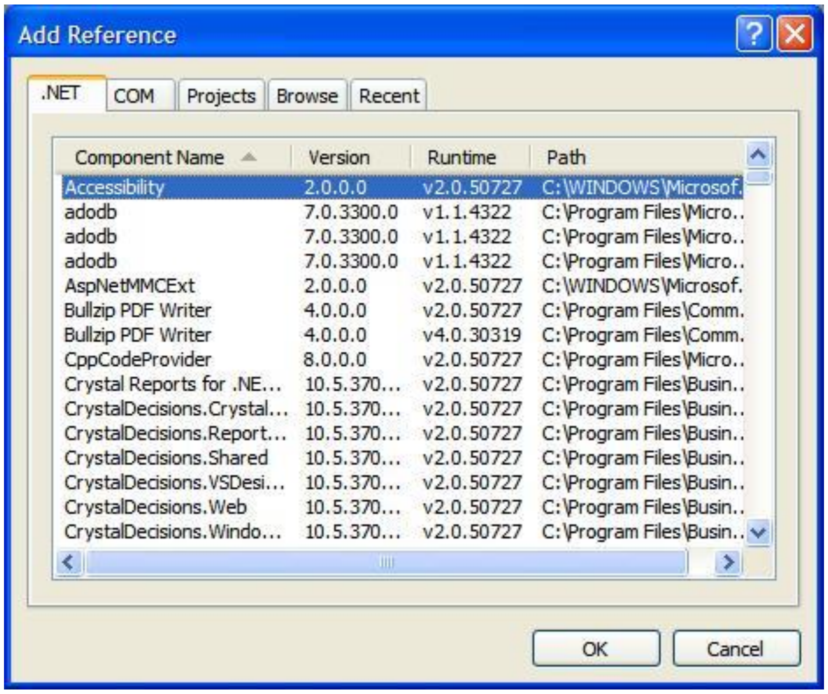
1. In the solution explorer right click on the project node and select "Add Reference".



You can also add the reference using the main menu. Navigate to "Project" and then select "Add Reference".

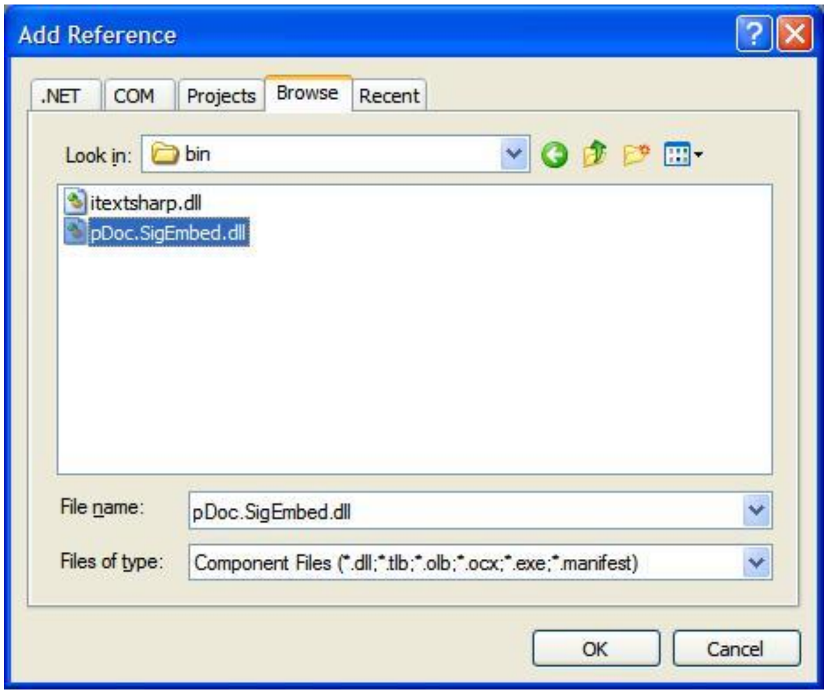


2. The following dialog will appear.



| Component Name | Version | Runtime | Path |
|---|---|---|---|
| Accessibility | 2.0.0.0 | v2.0.50727 | C:\WINDOWS\Microsof. |
| adodb | 7.0.3300.0 | v1.1.4322 | C:\Program Files\Micro.. |
| adodb | 7.0.3300.0 | v1.1.4322 | C:\Program Files\Micro.. |
| adodb | 7.0.3300.0 | v1.1.4322 | C:\Program Files\Micro.. |
| AspNetMMCExt | 2.0.0.0 | v2.0.50727 | C:\WINDOWS\Microsof. |
| Bullzip PDF Writer | 4.0.0.0 | v2.0.50727 | C:\Program Files\Comm. |
| Bullzip PDF Writer | 4.0.0.0 | v4.0.30319 | C:\Program Files\Comm. |
| CppCodeProvider | 8.0.0.0 | v2.0.50727 | C:\Program Files\Micro.. |
| Crystal Reports for .NE... | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.Crystal... | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.Report... | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.Shared | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.VSDesi... | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.Web | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |
| CrystalDecisions.Windo... | 10.5.370... | v2.0.50727 | C:\Program Files\Busin.. |

3. Select the "Browse" tab and navigate to the folder where the "pDoc.SigEmbed.dll" is located and select the assembly. Click "OK" to add the assembly to your project.



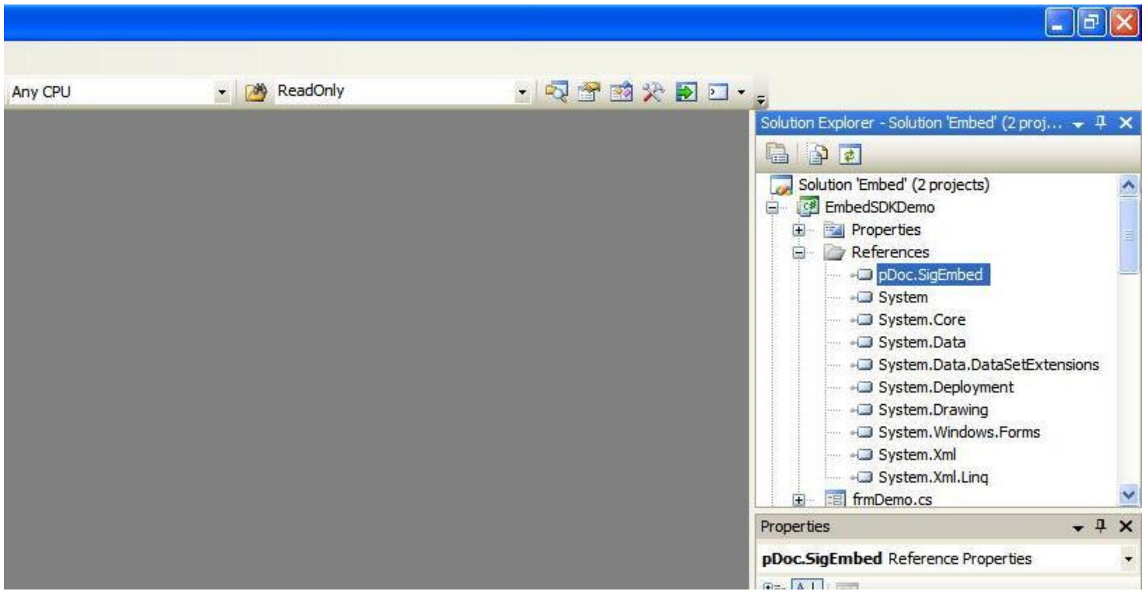
Look in: bin

itextsharp.dll
pDoc.SigEmbed.dll

File name: pDoc.SigEmbed.dll

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

4. The assembly will be added to the project as shown. Now you can start using the API exposed by the pDoc SigEmbed SDK.



## 4.0 – Namespaces

The API calls are encapsulated within the following namespaces.

- *Topaz.MultiPlatformSDK.Embed.PDFDocuments*

  This namespace will hold the classes responsible to manage the PDF document. These classes will encapsulate methods, properties, and fields required to manage the PDF document.

- *Topaz.MultiPlatformSDK.Embed.FormFields*

  This namespace will hold the classes responsible to manage form fields to the PDF document. These classes will encapsulate methods, properties, and fields required to manage form fields in a PDF document.

- *Topaz.MultiplatformSDK.Embed.PDFSigner*

  This namespace will hold the main class required for signing a PDF document and its supporting structures and enumerations. These classes will encapsulate methods, properties, and fields required to sign a signature field in a PDF document.

- *Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance*

  This namespace will hold the class responsible to create a signature appearance on a signature and its supporting structures and enumerations. These classes will encapsulate methods, properties, and fields required to create a signature appearance that gets displayed on the signature field.

- *Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries*

  This namespace will hold the class that can create custom dictionaries that can be added to a PDF signature dictionary. The namespace will also hold the supporting structures and enumerations.

- *Topaz.MultiPlatformSDK.Embed.PDFVerfier*

  This namespace will hold the main class required for verifying signatures and documents as well as the supporting structures and enumerations. These classes will encapsulate methods, properties, and fields required to verify signatures in a PDF document.

## 5.0 – Enumerations

### 5.1 The FieldTypes enumeration

Enumeration that lists the various field types.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFDocuments**

**Syntax**

```
public enum FieldTypes
{
Button = 1,
CheckBox = 2,
RadioButton = 3,
TextBox = 4,
ListBox = 5,
ComboBox = 6,
SignedSignature = 8,
UnsignedSignature = 9
}
```

## 5.2 The VisibilityOptions enumeration

Enumeration that lists the various visibility options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum VisibilityOptions
{
Visible = 0,
Hidden = 1,
VisiblePrintDisabled = 2,
HiddenPrintEnabled = 3
}
```

## 5.3 The OrientationOptions enumeration

Enumeration that lists the various orientation options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum OrientationOptions
{
ZeroDegrees = 0,
NinetyDegrees = 90,
OneEightyDegrees = 180,
TwoSeventyDegrees = 270
}
```

## 5.4 The BorderWidthOptions enumeration

Enumeration that lists the various border width options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum BorderWidthOptions
{
Thin = 1,
Medium = 2,
Thick = 3
}
```

## 5.5 The BorderStyles enumeration

Enumeration that lists the various border style options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum BorderStyles
{
Solid = 0,
Dashed = 1,
Beveled = 2,
Inset = 3,
Underlined = 4
}
```

## 5.6 The FontFaces enumeration

Enumeration that lists the various font faces available.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum FontFaces
{
NotSupported = 0,
Courier = 1,
CourierOblique = 2,
CourierBold = 3,
CourierBoldOblique = 4,
Helvetica = 5,
HelveticaOblique = 6,
HelveticaBold = 7,
HelveticaBoldOblique = 8,
TimesRoman = 9,
TimesItalic = 10,
TimesBold = 11,
TimesBoldItalic = 12,
Symbol = 13,
AdobePi = 14
}
```

## 5.7 The TextAlignOptions enumeration

Enumeration that lists the various text align options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum TextAlignOptions
{
Left = 0,
Center = 1,
Right = 2
}
```

## 5.8 The CheckStyles enumeration

Enumeration that lists the various check style options of the form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum CheckStyles
{
Check = 0,
Circle = 1,
Cross = 2,
Diamond = 3,
Square = 4,
Star = 5
}
```

## 5.9 The FieldListTypes enumeration

Enumeration that lists the field read only actions for signature fields.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public enum FieldListTypes
{
None = 0,
All = 1,
Exclude = 2,
Include = 3
}
```

## 5.10 The SignatureImageOptions enumeration

Enumeration that lists the various signature graphic placement options.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance**

**Syntax**

```
public enum SignatureImageOptions
{
ImageBesideDetails = 1,
ImageBehindDetails = 2,
ImageOnly = 3,
Automatic = 4
}
```

## 5.11 The StoreTypes enumeration

Enumeration that lists the available certificate store types.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**

```
public enum StoreTypes
{
PFX = 1
}
```

*Note: currently only PFX store based signing is supported.*

## 5.12 The SignatureActions enumeration

Enumeration that lists the available signature actions.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**

```
public enum SignatureActions : ushort
{
Sign = 1,
Initial = 2
}
```

## 5.13 The SignatureVerificationResults enumeration

Enumeration that lists the different signature verification results.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**
```
public enum SignatureVerificationResults
{
SignatureValidDocumentNotChanged = 1,
SignatureValidDocumentChanged = 2,
        SignatureInvalid = 3
}
```

## 5.14 The CertificateVerificationResults enumeration

Enumeration that lists the different certificate verification results.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**
```
public enum CertificateVerificationResults
{
        VerificationFailed = 0,
        VerificationPassed = 1
}
```

## 5.15 The CertificationLevels enumeration

Enumeration that lists the different certification levels.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**
```
public enum CertificationLevels
{
      CERTIFIED_NO_CHANGES_ALLOWED = 1,
      CERTIFIED_FORM_FILLING = 2,
      CERTIFIED_FORM_FILLING_AND_ANNOTATIONS = 3
}
```

# 6.0 – Interfaces

## 6.1 The IFormFieldMetadata interface

The base interface that defines the metadata of a form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public interface IFormFieldMetadata

### *6.1.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | FieldName | string FieldName{ get; set; } | The name of the form field. |
| | PageNumber | int PageNumber { get; set; } | The page number in the document where the form field is to be inserted. |
| | StartingX | float StartingX { get; set; } | The starting “X” co-ordinate of the form field in pixels with reference to user coordinate system i.e. top left as origin. |
| | StartingY | float StartingY { get; set; } | The starting “Y” co-ordinate of the form field in pixels with reference to user coordinate system i.e. top left as origin. |
| | Width | float Width { get; set; } | The width of the form field in pixels. |
| | Height | float Height { get; set; } | The height of the form field in pixels. |
| | Tooltip | string Tooltip { get; set; } | The tooltip for the form field. |
| | Visibility | VisibilityOptions Visibility { get; set; } | The visibility options of the form field. Possible values defined by the VisibilityOptions enumeration. |
| | Orientation | OrientationOptions Orientation { get; set; } | The orientation options of the form field. Possible values defined by the OrientationOptions enumeration. |
| | ReadOnly | bool ReadOnly { get; set; } | The read-only flag for the form field. |
| | Required | bool Required { get; set; } | The required flag for the form field. |
| | Locked | bool Locked { get; set; } | The locked flag for the form field. |

## 6.2 The IFormFieldStyle interface

The base interface that defines the style of a form field.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public interface IFormFieldStyle

### *6.2.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | BorderColor | Color BorderColor { get; set; } | The border color of the form field. |
| | BorderWidth | BorderWidthOptions BorderWidth { get; set; } | The border width of the form field. Possible values defined by the BorderWidthOptions enumeration. |
| | BorderStyle | BorderStyles BorderStyle { get; set; } | The border style of the form field. Possible values defined by the BorderStyles enumeration. |
| | FillColor | Color FillColor { get; set; } | The fill color of the form field. |
| | FontSize | ushort FontSize { get; set; } | The font size in a form field in points. |
| | FontColor | Color FontColor { get; set; } | The font color of the font in a form field. |
| | FontFace | FontFaces FontFace { get; set; } | The font face of the font in a form field. Possible values defined by the FontFaces enumeration. |

## 6.3 The IReadOnlyFormField interface

The base interface that defines form field read-only settings.
**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public interface IReadOnlyFormField

### 6.3.1 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | FieldListType | FieldListTypes FieldListType { get; set; } | The field list type. Possible values defined by the FieldListTypes enumeration. |
| | FieldList | List<string> FieldList { get; } | The list of field names to be included or excluded for Read-only settings. |
| | ErrorMessage | string ErrorMessage { get; } | The message that describes the last error. |

### 6.3.2 Methods

#### 6.3.2.1 The AddField method

**Method Description**
Adds a field to include or exclude for field read only after signing.

**Syntax**
bool AddField(string name);

**Parameters**
*name*
Type: string
The name that identifies the form field.

**Return Value**
A boolean value indicating if the form field was successfully added.

#### 6.3.2.2 The RemoveField method

**Method Description**
Removes a field included or excluded for field read only.

**Syntax**
bool RemoveField(string name);

**Parameters**
*name*
Type: string
The name that identifies the form field.

**Return Value**
A boolean value indicating if the form field was successfully removed.

**Usage**
This interface is exposed through the PDFSignField class. This interface allows you define the read-only form field settings. You can exclude or include form fields to be made read-only on signing. The settings defaults to none.

The following sample shows the usage. This information must be set before calling the method to add the signature field. For adding sample refer to the PDFSignField class.

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            ReadOnlyFieldSettings();
        }

        public static void ReadOnlyFieldSettings()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSignField class with PDFDocument object
            PDFSignField signField = new PDFSignField(document);
            // FieldListType can be None, All, Exclude, Include
            signField.ReadOnlyFormFields.FieldListType = FieldListTypes.Exclude;
                  // specify the field names for Exclude and Include
            // these fields will be made read-only when the signature field is
      signed
            signField.AddField("TextBox1");
            signField.AddField("TextBox2");
            signField.AddField("ComboBox1");
            signField.AddField("ComboBox2");
            signField.RemoveField("ComboBox1");
            // add the signature field
            ...
        }
    }
}
```

## 6.4 The IAppearanceInfo interface

The base interface that defines the appearance of a signature.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance**

**Syntax**
public interface IAppearanceInfo

### *6.4.1 Properties*

| | **Name** | **Syntax** | **Description** |
|---|---|---|---|
| | SignatureImage | Image SignatureImage { get; set; } | The signature image represented by an image object. |
| | SignatureData | byte[] SignatureData { get; set; } | The raw signature data from which the signature image to be shown on the appearance is to be created. |
| | SignatureImageOption | SignatureImageOptions SignatureImageOption { get; set; } | The signature image placement option. Possible values are ImageBesideDetails, ImageBehindDetails, ImageOnly, and Automatic as defined by the SignatureImageOptions enumeration. |
| | ShowSignerName | bool ShowSignerName { get; set; } | A flag indicating if the name of the person or authority signing the document should be shown in the appearance. The name can be specified using SignatureInfo property in PDFSign class. |
| | ShowReason | bool ShowReason { get; set; } | A flag indicating if the reason for signing should be shown in the appearance. The reason can be specified using SignatureInfo property in PDFSign class. |
| | ShowLocation | bool ShowLocation { get; set; } | A flag indicating if the host name or physical location of the signing should be shown in the appearance. The location can be specified using SignatureInfo property in PDFSign class. |
| | CustomText | string CustomText { get; set; } | The custom text to be displayed on the signature. If provided then the signature details on the signature will be replaced by the specified text. |
| | EnablepDocAppearance | bool EnablepDocAppearance { get; set; } | A flag indicating if the default pDoc Appearance should be used for displaying the details on the signature. In pDoc Appearance, the Signer Name (First Name and Last Name combined) and the Signature Date and Time are displayed on the signature. The font size is calculated based on the information available and the amount of space available for displaying the details. Setting this property ignores the custom text set through the CustomText property. |

**Usage**

This interface is exposed through the PDFSign class. This interface allows you define the appearance of the signature image and other details. The appearance defines what is visible to the end user. The following sample shows the usage. This information must be set before calling the method to sign. For signing sample refer to the PDFSign class.

```
using System;
using System.IO;
using System.Drawing;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        ...
        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            ...
            // define the appearance of the signature
            // set this to false if name
            // should not be displayed in the appearance
            sign.Appearance.ShowSignerName = true;
            // set this to false if reason
            // should not be displayed in the appearance
            sign.Appearance.ShowReason = true;
            // set the signature image
            sign.Appearance.SignatureImage = Image.FromFile(@"C:\Signature.jpg");
            // set the signature image options
            // options are ImageBesideDetails, ImageBehindDetails and ImageOnly
            // as defined by the SignatureImageOptions enumeration
            sign.Appearance.SignatureImageOption =
            SignatureImageOptions.ImageOnly;
            // use the custom text to overide the default appearance and
            // set a custom text of your choice
            // sign.Appearance.CustomText = "Signed by Topaz
            // MultiPlatform SDK\nDate:" + DateTime.Now;
            // sign the document
            ...
        }
    }
`
```

## 6.5 The ISignatureInfo interface

The base interface that specifies the signature information.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**

public interface ISignatureInfo

### *6.5.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | Filter | string Filter { get; } | The name of the signature handler to use when validating this signature. |
| | SubFilter | string SubFilter { get; } | The name that describes the encoding of the signature. |
| | FieldType | string FieldType { get; } | The type of PDF objects that the dictionary describes. |
| | SignDateTime | DateTime SignDateTime { get; } | The date and time of signing. |
| | SignDateTimeFormatted | string SignDateTimeFormatted { get; } | The date and time of signing in a custom format. This is used for display on the signature when the Appearance Type is set to “Automatic”. |
| | SignerName | string SignerName { get; set; } | The person or authority signing the document. |
| | Reason | string Reason { get; set; } | The reason for the signing. |
| | Location | string Location { get; set; } | The host name or physical location of the signing. |
| | ContactInfo | string ContactInfo { get; set; } | The contact information of the person or authority signing the document. |
| | Contents | string Contents { get; } | The contents entry of a signature dictionary. |
| | PKCS7 | byte[] PKCS7 { get; } | The encoded PKCS7 signature data. |
| | Reserved | string Reserved { set; } | Reserved parameter. |

**Usage**
This interface is exposed through the PDFSign class as well as the PDFVerify class. This interface allows you to set the name, reason for signing, and location of signing. The remaining information is read only and is available after the signing is performed.

The following sample shows the usage. This information must be set before calling the method to sign. The signature details can also be obtained using the GetSignatureInfo() method in the PDFVerify class.

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Sign();
        }

        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            sign.Signature.SignerName = "John Doe";
            sign.Signature.Reason = "Sample Testing";
            sign.Signature.Location = "Simi Valley";
            sign.Signature.ContactInfo = "test@test.com";
            // define the appearance of the signature
            ...

            // sign the document
            ...
        }
    }
}
```

## 6.6 The IStoreInfo interface

The base interface that specifies the store information.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**

public interface IStoreInfo

### *6.6.1 Properties*

| | **Name** | **Syntax** | **Description** |
|---|---|---|---|
| | StoreType | StoreTypes StoreType<br>{ get; set; } | The store where the certificate is located. Possible values defined by the StoreTypes enumeration. |
| | PFXData | byte[] PFXData<br>{ get; set; } | The raw PFX data. |
| | Password | string Password<br>{ get; set; } | The password to access the PFX store. |

**Usage**

This interface is exposed through the PDFSign class. This interface allows you to set the store information required to sign a PDF document. The following sample shows the usage.

This information must be set before calling the method to sign. For signing sample refer to the PDFSign class.

```
using System;
using System.IO;
using System.Drawing;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Sign();
        }

        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            ...
            // define the appearance of the signature
            ...
            // set the store data
            // presently only PFX store type supported
            sign.Store.StoreType = StoreTypes.PFX;
            // read the PFX data
            sign.Store.PFXData = File.ReadAllBytes(@"C:\Sign.pfx");
            // set the password
            sign.Store.Password = "password";
            // sign the document
            ...
        }
    }
}
```

## 6.7 The ICertificateInfo interface

The base interface that specifies the certificate information.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**
public interface ICertificateInfo

### *6.7.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | CertificateData | byte[] CertificateData { get; set; } | The raw certificate data. |
| | CertificationLevel | `CertificationLevels CertificationLevel`{ get; set; } | The certification level which specifies what changes are allowed on the certified document. Allowed options are “No Changes”, “Form Field Filling”, and “Form Field Filling and Annotations”. |

**Usage**

This interface is exposed through the PDFSign class. This interface allows you to set the certificate for two pass distributed signing (PreSign() and PostSign() methods). The following sample shows the usage. This information must be set before calling the methods to sign. For signing sample refer to the PDFSign class.

```
using System;
using System.IO;
using System.Drawing;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;
namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Sign();
        }
        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            ...
            // set define the appearance of the signature
            ...
            // set the certificate
            sign.Certificate.CertificateData = File.ReadAllBytes(@"C:\Sign.cer");
            // set the certification level
            sign.Certificate.CertificationLevel =
            CertificationLevels.CERTIFIED_FORM_FILLING;
            // sign the document
        }
```

## 6.8 The ICertificateDetails interface

The base interface that exposes the certificate details.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**

public interface ICertificateDetails

### *6.8.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | IssuedTo | string IssuedTo { get; } | The name of the person or entity to which the certificate was issued. |
| | Issuer | string Issuer { get; } | The name of the certification authority that issued the certificate. |
| | Thumbprint | string Thumbprint { get; } | A hexadecimal string that contains the SHA-1 hash of the certificate. |
| | BeginDate | DateTime BeginDate { get; } | The beginning date for the validity of the certificate. |
| | EndDate | DateTime EndDate { get; } | The ending date for the validity of the certificate. |
| | SerialNumber | string SerialNumber { get; } | The certificate serial number. |
| | VersionNumber | int VersionNumber { get; } | The version number of the certificate. |
| | PublicKey | string PublicKey { get; } | The public key for the certificate. |
| | PublicKeyAlgorithm | string PublicKeyAlgorithm { get; } | The key algorithm information for the certificate. |
| | SignatureAlgorithm | string SignatureAlgorithm { get; } | The signature algorithm information for the certificate. |
| | RawCertificateData | byte[] RawCertificateData { get; } | The raw data for the X.509v3 certificate. |

## 6.9 The ISignatureVerificationDetails interface

The base interface that exposes the signature verification details.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**

public interface ISignatureVerificationDetails

### *6.9.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | Result | SignatureVerificationResults Result { get; } | The actual result of the signature verification defined by the SignatureVerificationResults enumeration. |
| | Message | string Message { get; } | The signature verification result message. |
| | Revision | int Revision { get; } | The document revision covered by the signature. |
| | TotalRevisions | int TotalRevisions { get; } | The total number of revisions in the document. |

## 6.10 The ICertificateVerificationDetails interface

The base interface that exposes the certificate verification details.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**

public interface ICertificateVerificationDetails

### *6.10.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | Result | CertificateVerificationResults Result { get; } | The actual result of the certificate verification defined by the CertificateVerificationResults enumeration. |
| | Message | string Message { get; } | The certificate verification result message. |

# 7.0 – Classes

## 7.1 The PDFDocument class

The main class for managing a PDF document.

**Namespace**

**Topaz.MultiPlatformSDK.Embed.PDFDocuments**

**Syntax**

```
public class PDFDocument : IDisposable
```

### *7.1.1 Constructors*

**Syntax**

```
public PDFDocument(byte[] documentBytes)
```

**Parameters**

*documentBytes*
Type: byte array.
The PDF document bytes.

**Syntax**

```
public PDFDocument(byte[] documentBytes, string openPassword)
```

**Parameters**

*documentBytes*
Type: byte array
The PDF document bytes.
*openPassword*
Type: string
The password to open the document.

*Note: Both constructors throw a **UnauthorizedAccessException** if the document is secured and unable to open.*

## 7.1.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | IsDocumentRestricted | public bool IsDocumentRestricted { get; } | A flag indicating if the document is restricted with permissions password. |
| | PrintingAllowed | public bool PrintingAllowed { get; } | A flag indicating if printing is allowed for the document. |
| | DocumentChangesAllowed | public bool DocumentChangesAllowed { get; } | A flag indicating if changes to the document are allowed. |
| | DocumentAssemblyAllowed | public bool DocumentAssemblyAllowed { get; } | A flag indicating if content copying is allowed for the document. |
| | ContentCopyingAllowed | public bool ContentCopyingAllowed { get; } | A flag indicating if content copying is allowed for the document. |
| | PageExtractionAllowed | public bool PageExtractionAllowed { get; } | A flag indicating if extraction of pages allowed from the document. |
| | CommentingAllowed | public bool CommentingAllowed { get; } | A flag indicating if commenting is allowed in the document. |
| | FormFieldFillingAllowed | public bool FormFieldFillingAllowed { get; } | A flag indicating if form filling is allowed for the document. |
| | SigningAllowed | public bool SigningAllowed { get; } | A flag indicating if signing is allowed for the document. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

## 7.1.3 Methods

### 7.1.3.1 The GetFieldList method

**Method Description**
Gets the list of field names according to type in the PDF document.

**Syntax**
```
public List<string> GetFieldList(FieldTypes fieldType)
```

**Parameters**
*fieldType*
Type: FieldTypes enumeration.
The type of field whose list is to be retrieved.

**Return Value**
The list of field names. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetFieldList();
        }

        public static void GetFieldList()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            List<string> formFields = null;
            // Get the list of CheckBox field names
            //formFields = document.GetFieldList(FieldTypes.CheckBox);
            // Get the list of ComboBox field names
            //formFields = document.GetFieldList(FieldTypes.ComboBox);
            // Get the list of ListBox field names
            //formFields = document.GetFieldList(FieldTypes.ListBox);
            // Get the list of RadioButton field names
            formFields = document.GetFieldList(FieldTypes.RadioButton);
            // Get the list of signed signature field names
            //formFields = document.GetFieldList(FieldTypes.SignedSignature);
            // Get the list of textbox field names
            //formFields = document.GetFieldList(FieldTypes.TextBox);
            // Get the list of unsigned signature field names
            formFields = document.GetFieldList(FieldTypes.UnsignedSignature);
            if (formFields != null)
            {
                // loop through form fields
                ...
            }
            else
                Console.WriteLine("Error: {0}", document.ErrorMessage);
        }
    }
}
```

### 7.1.3.2 The GetBytes method

**Method Description**
Gets the currently saved PDF document bytes.

**Syntax**
public byte[] GetBytes()

**Parameters**
*None.*

**Return Value**
The currently saved PDF document bytes.

*Note: Should be invoked to get the PDF document bytes after manipulation (i.e. Adding a filed, deleting a field, updating a filed with value, changing the state of the field, signing etc...).*

**Usage**

```
using System;
using System.IO;
using System.Drawing;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DoSomething();
        }

        public static void DoSomething()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
             // initialize the PDFDocument class with input document
             PDFDocument document = new PDFDocument(inputDocument);
             // do something - sign, create fields, fill etc.
            ...
             // get the document bytes and save
             byte[] outputDocument = document.GetBytes();
             if (outputDocument != null)
            {
                // write the output document to the file system
                ...
            }
            else
                Console.WriteLine("Error: {0}", document.ErrorMessage);

             // always close the document after the job is done
             // close the document
             document.Close();
        }
    }
}
```

#### 7.1.3.3 The Close method

**Method Description**
Closes the document and releases all the resources associated with it.

**Syntax**
public void Close()

**Parameters**
*None.*

**Return Value**
None.

#### 7.1.3.4 The Dispose method

**Method Description**
IDisposable Implementation.

**Syntax**
public void Dispose()

**Parameters**
*None.*

**Return Value**
None.

## 7.2 The PDFFormField class

The base abstract class for managing PDF form fields.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public abstract class PDFFormField

### 7.2.1 Constructors

**Syntax**
protected PDFFormField()

**Parameters**
None.

### 7.2.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | Metadata | public IFormFieldMetadata Metadata { get; } | The form field metadata. Exposed through IFormFieldMetadata interface. |
| | Style | public IFormFieldStyle Style { get; } | The form field style. Exposed through IFormFieldStyle interface. |

### 7.2.3 Methods

#### 7.2.3.1 The Add method

**Method Description**
Adds a form field to a PDF document.

**Syntax**
public abstract bool Add();

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the form field was successfully added.

#### 7.2.3.2 The Delete method

**Method Description**
Deletes a form field from a PDF Document.

**Syntax**
public abstract bool Delete();

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the form field was successfully deleted.

## 7.3 The PDFTextBoxField class

Class for managing PDF TextBox fields. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public class PDFTextBoxField : PDFFormField

### 7.3.1 Constructors

**Syntax**
public PDFTextBoxField(PDFDocument pdfDocument) : base()

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### 7.3.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | DefaultValue | public string DefaultValue { get; set; } | The default value of the TextBox field. |
| | TextAlign | public TextAlignOptions TextAlign { get; set; } | The text alignment in a TextBox field. Possible values defined by the TextAlignOptions enumeration. |
| | IsPasswordField | public bool IsPasswordField { get; set; } | A flag indicating if the TextBox is a password field. |
| | IsMultiline | public bool IsMultiline { get; set; } | A flag indicating if the TextBox field is multiline. |
| | CharacterLimit | public uint CharacterLimit { get; set; } | The character limit of the TextBox field. |
| | ScrollText | public bool ScrollText { get; set; } | A flag indicating if text should be scrolled in the TextBox field. |
| | CombCharacters | public uint CombCharacters { get; set; } | Number of comb characters in a TextBox field. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### 7.3.3 Methods

#### 7.3.3.1 The Add method

**Method Description**
Adds a TextBox field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Add()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the TextBox field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFTextBoxField class with the PDFDocument object
            PDFTextBoxField textBoxField = new PDFTextBoxField(document);
            // set the PDFTextBoxField Metadata parameters
            textBoxField.Metadata.StartingX = 100;
            textBoxField.Metadata.StartingY = 100;
            textBoxField.Metadata.Height = 150;
            textBoxField.Metadata.Width = 300;
            textBoxField.Metadata.PageNumber = 1;
            textBoxField.Metadata.Tooltip = "TextBox1";
            textBoxField.Metadata.FieldName = "TextBox1";
            textBoxField.Metadata.Visibility = VisibilityOptions.Visible;
            textBoxField.Metadata.Orientation = OrientationOptions.ZeroDegrees;
            // set the PDFTextBoxField Style parameters
            textBoxField.Style.BorderColor = System.Drawing.Color.Empty;
            textBoxField.Style.BorderWidth = BorderWidthOptions.Thin;
            textBoxField.Style.FillColor =  System.Drawing.Color.Empty;
            textBoxField.Style.FontColor =  System.Drawing.Color.Empty;
            textBoxField.Style.FontFace = FontFaces.Helvetica;
            textBoxField.Style.FontSize = 0;
            textBoxField.Style.BorderStyle = BorderStyles.Solid;
            // set the PDFTextBoxField parameters
            textBoxField.DefaultValue = " ";
            textBoxField.TextAlign = TextAlignOptions.Left;
            textBoxField.IsPasswordField = false;
            textBoxField.IsMultiline = false;
            textBoxField.CharacterLimit = 0;
            textBoxField.ScrollText = false;
            textBoxField.CombCharacters = 0;
            // add the TextBox field
            textBoxField.Add();
            // get the document bytes and save
            ...
        }
    }
}
```

### 7.3.3.2 The Delete method

**Method Description**
Deletes a TextBox field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Delete()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the TextBox field was successfully deleted.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFTextBoxField class with the PDFDocument object
            PDFTextBoxField  textBoxField = new PDFTextBoxField(document);
            // delete the TextBox field
            textBoxField.Metadata.FieldName = "TextBox1";
            bool result = textBoxField.Delete();
            if (result)
            {
                Console.WriteLine("TextBox field deleted successfully.");
               // get the document bytes and save
               ...
            }
            else
                Console.WriteLine("Error: {0}", textBoxField.ErrorMessage;

       }
    }
}
```

### 7.3.3.3 The Fill method

**Method Description**
Fills the TextBox field with the value specified.

**Syntax**
public bool Fill(string fieldValue)

**Parameters**
*fieldValue*
Type: string
The TextBox field value.

**Return Value**
A boolean value indicating if the value was successfully filled.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            FillField();
        }

        public static void FillField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFTextBoxField class with PDFDocument object
            PDFTextBoxField  textBoxField = new PDFTextBoxField(document);
            // fill the TextBox field
            textBoxField.Metadata.FieldName = "TextBox1";
            string value = "Text1";
            bool result = textBoxField.Fill(value);
            if (result)
            {
                Console.WriteLine("TextBox field filled successfully.");
                // get the document bytes and save
                ...
            }
            else
                Console.WriteLine("Error: {0}", textBoxField.ErrorMessage);
        }
    }
}
```

#### 7.3.3.4 The GetValue method

**Method Description**
Gets the value of TextBox field.

**Syntax**
public string GetValue()

**Parameters**
*None.*

**Return Value**
The value of TextBox field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetValue();
        }

        public static void GetValue()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFTextBoxField class with the PDFDocument object
            PDFTextBoxField  textBoxField = new PDFTextBoxField(document);
            textBoxField.Metadata.FieldName = "TextBox1";
            string textVal = textBoxField.GetValue();
            if (textVal != null)
                Console.WriteLine("Value: {0}", textVal);
            else
                Console.WriteLine("Error: {0}", textBoxField.ErrorMessage);
        }
    }
}
```

## 7.4 The PDFComboBoxField class

Class for managing PDF ComboBox fields. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public class PDFComboBoxField : PDFFormField

### 7.4.1 Constructors

**Syntax**
public PDFComboBoxField(PDFDocument pdfDocument) : base()

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### 7.4.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | DefaultSelection | public string DefaultSelection { get; set; } | The default selected value in the ComboBox field. |
| | Editable | public bool Editable { get; set; } | A flag indicating if the ComboBox field is editable |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### 7.4.3 Methods

#### *7.4.3.1 The AddItem method*

**Method Description**
Adds an item to the ComboBox field. Should be called before the control is added.

**Syntax**
public bool AddItem(string itemText, string exportValue)

**Parameters**
*itemText*
Type: string
The text that identifies the item to be added.
*exportValue*
Type: string
The export value to be set against the item.

**Return Value**
A boolean value indicating if the item was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddItem();
        }

        public static void AddItem()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFComboBoxField class with the PDFDocument object
            PDFComboBoxField  comboBoxField = new PDFComboBoxField(document);
            comboBoxField.Metadata.FieldName = "ComboBox1";
            // add the item and the export value
            string comboItem = "Item1";
            string comboExportVal = "Value1";
            bool result = comboBoxField.AddItem(comboItem, comboExportVal);
            if (result)
                Console.WriteLine("Item added.");
            else
                Console.WriteLine("Error: {0}", comboBoxField.ErrorMessage);
        }
    }
}
```

### 7.4.3.2 The RemoveItem Method

**Method Description**
Removes an item from the ComboBox field. Should be called before the control is added.

**Syntax**
public bool RemoveItem(string itemText)

**Parameters**
*itemText*
Type: string
The text that identifies the item to be removed.
**Return Value**
A boolean value indicating if the item was successfully removed.

**Usage**
```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            RemoveItem();
        }

        public static void RemoveItem()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFComboBoxField class with the PDFDocument object
            PDFComboBoxField  comboBoxField = new PDFComboBoxField(document);
            // remove the item
            comboBoxField.Metadata.FieldName = "ComboBox1";
            string comboItem = "Item1";
            bool result = comboBoxField.RemoveItem(comboItem);
            if (result)
                Console.WriteLine("Item removed.");
            else
                Console.WriteLine("Error: {0}", comboBoxField.ErrorMessage);
        }
    }
}
```

### *7.4.3.3 The Add method*

**Method Description**
Adds a ComboBox field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Add()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the ComboBox field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFComboBoxField class with the PDFDocument object
            PDFComboBoxField comboBoxField = new DFComboBoxField(document);
            // set the PDFComboBoxfield Metadata parameters
            comboBoxField.Metadata.StartingX = 100;
            comboBoxField.Metadata.StartingY = 100;
            comboBoxField.Metadata.Height = 150;
            comboBoxField.Metadata.Width = 300;
            comboBoxField.Metadata.PageNumber = 1;
            comboBoxField.Metadata.Tooltip = "ComboBox1";
            comboBoxField.Metadata.FieldName = "ComboBox1";
            comboBoxField.Metadata.Visibility = VisibilityOptions.Visible;
            comboBoxField.Metadata.Orientation = OrientationOptions.ZeroDegrees;
            comboBoxField.Metadata.ReadOnly = false;
            comboBoxField.Metadata.Required = false;
            comboBoxField.Metadata.Locked = false;
            // set the PDFComboBoxfield Style parameters
            comboBoxField.Style.BorderColor = System.Drawing.Color.Empty;
            comboBoxField.Style.BorderWidth = BorderWidthOptions.Thin;
            comboBoxField.Style.FillColor =  System.Drawing.Color.Empty;
            comboBoxField.Style.FontColor =  System.Drawing.Color.Empty;
            comboBoxField.Style.FontFace = FontFaces.Helvetica;
            comboBoxField.Style.FontSize = 0;
            comboBoxField.Style.BorderStyle = BorderStyles.Solid;
            comboBoxField.Editable = false;
            comboBoxField.DefaultSelection = "Item1";
            // add the ComboBox field
            comboBoxField.Add();
            ...
        }
    }
}
```

#### 7.4.3.4 The Delete method

**Method Description**
Deletes a ComboBox field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
```
public override bool Delete()
```

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the ComboBox field was successfully deleted.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFComboBox class with PDFDocument object
            PDFComboBoxField  comboBoxField = new DFComboBoxField(document);
            // delete the ComboBox field
            comboBoxField.Metadata.FieldName = "ComboBox1";
            bool result = comboBoxField.Delete();
            if (result)
                Console.WriteLine("ComboBox field deleted successfully.");
            else
                Console.WriteLine("Error: {0}", comboBoxField.ErrorMessage);

        }
    }
}
```

### 7.4.3.5 The SelectItem method

**Method Description**
Selects an item in the ComboBox field. Should be called after the control is added.

**Syntax**
public bool SelectItem(string itemText, string exportValue)

**Parameters**
*itemText*
Type: string
The text that identifies the item to be selected.
*exportValue*
Type: string
The export value of the item to be selected.

**Return Value**
A boolean value indicating if the item was successfully selected.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            FillField();
        }

        public static void FillField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFComboBox Field class with the PDFDocument object
            PDFComboBoxField comboBoxField = new PDFComboBoxField(document);
            comboBoxField.Metadata.FieldName = "ComboBox1";
            string comboItem = "Item1";
            string comboExportVal = "Value1";
            bool result =  comboBoxField.SelectItem(comboItem, comboExportVal);
            if (result)
                Console.WriteLine("ComboBox field filled successfully.");
            else
            Console.WriteLine("Error: {0}", comboBoxField.ErrorMessage);
        }
    }
}
```

### 7.4.3.6 The GetItems method

**Method Description**
Gets the list of items in the form of key value pairs.

**Syntax**
public Dictionary<string, string> GetItems()

**Parameters**
*None.*

**Return Value**
The list of items in the form of key value pairs.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetItems();
        }

        public static void GetItems()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFComboBox Field class with the PDFDocument object
            PDFComboBoxField comboBoxField = new PDFComboBoxField(document);
            // get the items in the ComboBox field
            comboBoxField.Metadata.FieldName = "ComboBox1";
            Dictionary<string, string> comboItemDic = comboBoxField.GetItems();
            // loop through the items
            ...
        }
    }
}
```

### 7.4.3.7 The GetSelectedItem method

**Method Description**
Gets the selected item in the ComboBox field.

**Syntax**
public string GetSelectedItem()

**Parameters**
*None.*

**Return Value**
The selected item in the ComboBox field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetSelectedItem();
        }

        public static void GetSelectedItem()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFComboBoxField class with the formFillingDocument
            PDFComboBoxField comboBoxField = new PDFComboBoxField(document);
            // get the selected item
            comboBoxField.Metadata.FieldName = "ComboBox1";
            // the selected item returns the export value
            string itemExportVal =  comboBoxField.GetSelectedItem();
            if (itemExportVal != null)
                Console.WriteLine("Export Value: {0}", itemExportVal);
            else
                Console.WriteLine("Error: {0}", comboBoxField.ErrorMessage);

        }
    }
}
```

## 7.5 The PDFListBoxField class

Class for managing PDF ListBox fields. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**

```
public class PDFListBoxField : PDFFormField
```

### 7.5.1 Constructors

**Syntax**

```
public PDFListBoxField(PDFDocument pdfDocument) : base()
```

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### 7.5.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | DefaultSelection | public string DefaultSelection { get; set; } | The default selected value in the ListBox field. |
| | AllowMultipleSelection | public bool AllowMultipleSelection { get; set; } | A flag indicating if multiple selections are allowed in the ListBox field. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### 7.5.3 Methods

### 7.5.3.1 The AddItem method

**Method Description**
Adds an item to the ListBox field. Should be called before the control is added.

**Syntax**
public bool AddItem(string itemText, string exportValue)

**Parameters**
*itemText*
Type: string
The text that identifies the item to be added.
*exportValue*
Type: string
The export value to be set against the item.

**Return Value**
A boolean value indicating if the item was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddItem();
        }

        public static void AddItem()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFListBoxField class with the PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            // add the item to the ListBox field
            listBoxField.Metadata.FieldName = "ListBox1";
            string listItemVal = "Item1";
            string listExportVal = "Value1";
            bool result = listBoxField.AddItem(listItemVal, listExportVal);
            if (result)
                Console.WriteLine("Item added.");
            else
                Console.WriteLine("Error: {0}", listBoxField.ErrorMessage);
        }
    }
}
```

### 7.5.3.2 The RemoveItem method

**Method Description**
Removes an item from the ListBox field. Should be called before the control is added.

**Syntax**
public bool RemoveItem(string itemText)

**Parameters**
*itemText*
Type: string
The text that identifies the item to be removed.
**Return Value**
A boolean value indicating if the item was successfully removed.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            RemoveItem();
        }

        public static void RemoveItem()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFListBoxField class with PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            // remove the item from the ListBox field
            listBoxField.Metadata.FieldName = "ListBox1";
            string listItemVal = "Item1";
            bool result = listBoxField.RemoveItem(listItemVal);
            if (result)
                Console.WriteLine("Item removed.");
            else
                Console.WriteLine("Error: {0}", listBoxField.ErrorMessage);
        }
    }
}
```

### 7.5.3.3 The Add method

**Method Description**

Adds a ListBox field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**

```
public override bool Add()
```

**Parameters**

*None.*

**Return Value**

A boolean value indicating if the ListBox field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(document);
            // initialize the PDFListBox class with the PDFDocument object
            PDFListBoxField listBoxField = new PDFListBoxField(document);
            // set the PDFListBox field Metadata parameters
            listBoxField.Metadata.StartingX = 100;
            listBoxField.Metadata.StartingY = 100;
            listBoxField.Metadata.Height = 150;
            listBoxField.Metadata.Width = 300;
            listBoxField.Metadata.PageNumber = 1;
            listBoxField.Metadata.Tooltip = "ListBox1";
            listBoxField.Metadata.FieldName = "ListBox1";
            listBoxField.Metadata.Visibility = VisibilityOptions.Visible;
            listBoxField.Metadata.Orientation = OrientationOptions.ZeroDegrees;
            listBoxField.Metadata.ReadOnly = false;
            listBoxField.Metadata.Locked = false;
            // set the PDFListBox field style parameters
            listBoxField.Style.BorderColor = System.Drawing.Color.Empty;
            listBoxField.Style.BorderWidth = BorderWidthOptions.Thin;
            listBoxField.Style.FillColor =  System.Drawing.Color.Empty;
            listBoxField.Style.FontColor =  System.Drawing.Color.Empty;
            listBoxField.Style.FontFace = FontFaces.Helvetica;
            listBoxField.Style.FontSize = 0;
            listBoxField.Style.BorderStyle = BorderStyles.Solid;
            listBoxField.AllowMultipleSelection = false;
            listBoxField.DefaultSelection = "Item1";
            // add the ListBox field
            listBoxField.Add();
            ...
        }
    }
}
```

### 7.5.3.4 The Delete method

**Method Description**
Deletes a ListBox field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Delete()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the ListBox field was successfully deleted.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFListBox class with the PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            // delete the ListBox field
            listBoxField.Metadata.FieldName = "ListBox1";
            bool result = listBoxField.Delete();
            if (result)
                Console.WriteLine("ListBox field deleted successfully.");
            else
                Console.WriteLine("Error: {0}", listBoxField .ErrorMessage);

        }
    }
}
```

### 7.5.3.5 The SelectItems method

**Method Description**
Selects the list of items in a ListBox field.

**Syntax**
public bool SelectItems(Dictionary<string, string> items)

**Parameters**
*items*
Type: Dictionary object.
A dictionary of items with item name and export value.

**Return Value**
A boolean value indicating if the list of items were successfully selected in the ListBox field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            FillField();
        }

        public static void FillField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFListBoxField class with PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            listBoxField.Metadata.FieldName = "ListBox1";
            // for multiple selection add multiple items
            // ListBox should allow multiple selection
            Dictionary<string, string> selItemDic= new Dictionary<string,
            string>();
            selItemDic.Add("Item1", "Value1");
            selItemDic.Add("Item2", "Value2");
            bool result = listBoxField.SelectItems(selItemDic);
            if (result)
                Console.WriteLine("ListBox field filled successfully.");
            else
                Console.WriteLine("Error: {0}", listBoxField.ErrorMessage);
        }
    }
}
```

### 7.5.3.6 The GetItems method

**Method Description**
Gets the list of items in the form of key value pairs.

**Syntax**
public Dictionary<string, string> GetItems()

**Parameters**
*None.*

**Return Value**
The list of items in the form of key value pairs.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetItems();
        }

        public static void GetItems()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFListBoxField class with PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            // get the items in a ListBox field
            listBoxField.Metadata.FieldName = "ListBox1";
            Dictionary<string, string> listItemDic = listBoxField.GetItems();
            // loop through the items
            ...
        }
    }
}
```

### 7.5.3.7 The GetSelectedItems method

**Method Description**
Gets a list of selected items in a ListBox field.

**Syntax**
public string[] GetSelectedItems()

**Parameters**
*None.*

**Return Value**
A list of selected items in a ListBox field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetSelectedItems();
        }

        public static void GetSelectedItems()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFListBoxField class with the PDFDocument object
            PDFListBoxField  listBoxField = new PDFListBoxField(document);
            // get the selected items in a ListBox field
            listBoxField.Metadata.FieldName = "ListBox1";
            Dictionary<string, string> dic = listBoxField .GetSelectedItems();
            // loop through the selected items
            ...
        }
    }
}
```

## 7.6 The PDFCheckBoxField class

Class for managing PDF CheckBox fields. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public class PDFCheckBoxField : PDFFormField

### *7.6.1 Constructors*

**Syntax**
public PDFCheckBoxField(PDFDocument pdfDocument) : base()

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### *7.6.2 Properties*

| | **Name** | **Syntax** | **Description** |
|---|---|---|---|
| | ExportValue | public string ExportValue { get; set; } | The export value of the CheckBox field. |
| | CheckStyle | public CheckStyles CheckStyle { get; set; } | The various check style options available Defined by the CheckStyles enumeration. |
| | CheckedByDefault | public bool CheckedByDefault { get; set; } | A flag indicating if the CheckBox field is checked by default. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### *7.6.3 Methods*

#### *7.6.3.1 The Add method*

**Method Description**
Adds a CheckBox field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Add()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the CheckBox field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFCheckBoxField class with PDFDocument object
            PDFCheckBoxField  checkBoxField = new PDFCheckBoxField(document);
            // set the PDFCheckBox field Metadata parameters
            checkBoxField.Metadata.StartingX = 100;
            checkBoxField.Metadata.StartingY = 100;
            checkBoxField.Metadata.Height = 150;
            checkBoxField.Metadata.Width = 300;
            checkBoxField.Metadata.PageNumber = 1;
            checkBoxField.Metadata.Tooltip = "CheckBox1";
            checkBoxField.Metadata.FieldName = "CheckBox1";
            checkBoxField.Metadata.Visibility = VisibilityOptions.Visible;
            checkBoxField.Metadata.Orientation = OrientationOptions.ZeroDegrees;
            checkBoxField.Metadata.ReadOnly = false;
        checkBoxField.Metadata.Required = false;
        checkBoxField.Metadata.Locked = false;
             // set the PDFCheckBox field style parameters
             checkBoxField.Style.BorderColor = System.Drawing.Color.Empty;
            checkBoxField.Style.BorderWidth = BorderWidthOptions.Thin;
            checkBoxField.Style.FillColor =  System.Drawing.Color.Empty;
            checkBoxField.Style.FontColor =  System.Drawing.Color.Empty;
            checkBoxField.Style.FontFace = FontFaces.Helvetica;
            checkBoxField.Style.FontSize = 0;
            checkBoxField.Style.BorderStyle = BorderStyles.Solid;
            checkBoxField.CheckedByDefault = false;
             checkBoxField.ExportValue = "Yes";
            checkBoxField.CheckStyle = CheckStyles.Check;
             // add the CheckBox field
             bool result = checkBoxField.Add();
             if (result)
                Console.WriteLine("CheckBox field added successfully.");
            else
                Console.WriteLine("Error: {0}", checkBoxField.ErrorMessage);
        }
    }
}
```

### 7.6.3.2 The Delete method

**Method Description**
Deletes a CheckBox field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Delete()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the CheckBox field was successfully deleted.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFCheckBoxField class with PDFDocument object
            PDFCheckBoxField  checkBoxField = new PDFCheckBoxField(document);
            // delete the CheckBox field
            checkBoxField.Metadata.FieldName = "CheckBox1";
            bool result = checkBoxField .Delete();
            if (result)
                Console.WriteLine("CheckBox field deleted successfully.");
            else
                Console.WriteLine("Error: {0}", checkBoxField.ErrorMessage);
        }
    }
}
```

#### 7.6.3.3 The SetState method

**Method Description**
Sets the state of the specified CheckBox field.

**Syntax**
public bool SetState(string exportValue)

**Parameters**
*exportValue*
Type: string
The export value of the CheckBox field.

**Return Value**
A boolean value indicating if the state was successfully set.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            FillField();
        }

        public static void FillField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFCheckBoxField class with the PDFDocument object
            PDFCheckBoxField  checkBoxField = new PDFCheckBoxField(document);
            // set the CheckBox field state (check)
            // to uncheck set the state with export value as “Off”
            checkBoxField.Metadata.FieldName = "CheckBox1";
            string exportValue = checkBoxField.GetValue();
            bool result = checkBoxField.SetState(exportValue);
            if (result)
                Console.WriteLine("CheckBox Field state set successfully.");
            else
                Console.WriteLine("Error: {0}", checkBoxField.ErrorMessage);
        }
    }
}
```

### 7.6.3.4 The GetStates method

**Method Description**
Gets the states of the specified CheckBox field.

**Syntax**
public List<string> GetStates()

**Parameters**
None.

**Return Value**
All of the possible export values of the CheckBox.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetStates();
        }

        public static void GetStates()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFCheckBoxField class with PDFDocument object
            PDFCheckBoxField  checkBoxField = new PDFCheckBoxField(document);
            // get the export values of the CheckBox field
            checkBoxField.Metadata.FieldName = "CheckBox1";
            List<string> exportValues = checkBoxField.GetStates();
            // loop through the export values
            ...
        }
    }
}
```

### 7.6.3.5 The GetValue method

**Method Description**
Gets the export value of CheckBox field.

**Syntax**
public string GetValue()

**Parameters**
*None.*

**Return Value**
The current selected export value of CheckBox field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetValue();
        }

        public static void GetValue()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFCheckBoxField class with PDFDocument object
            PDFCheckBoxField  checkBoxField = new PDFCheckBoxField(document);
            // get the export value of the CheckBox field
            checkBoxField.Metadata.FieldName = "CheckBox1";
            string exportValue = checkBoxField.GetValue();
            if (exportValue != null)
                Console.WriteLine("Export Value: {0}", exportValue);
            else
                Console.WriteLine("Error: {0}", checkBoxField.ErrorMessage);

        }
    }
}
```

## 7.7 The PDFRadioButtonFieldclass

Class for managing PDF RadioButton fields. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public class PDFRadioButtonField: PDFFormField

### *7.7.1 Constructors*

**Syntax**
public PDFRadioButtonField(PDFDocument pdfDocument) : base()

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### *7.7.2 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | ExportValue | public string ExportValue<br>{ get; set; } | The export value of the RadioButton field. |
| | ExportIndex | public uint ExportIndex<br>{ get; set; } | The export index of the RadioButton field. |
| | CheckStyle | public CheckStyles CheckStyle<br>{ get; set; } | The various check style options available Defined by the CheckStyles enumeration. |
| | CheckedByDefault | public bool CheckedByDefault<br>{ get; set; } | A flag indicating if the RadioButton field is checked by default. |
| | Unison | public bool Unison<br>{ get; set; } | A flag indicating if the RadioButton field is in in unison with other fields with the same name. |
| | ErrorMessage | public string ErrorMessage<br>{ get; } | Gets the message that describes the last error. |

### 7.7.3 Methods

#### 7.7.3.1 The Add method

**Method Description**
Adds a RadioButton field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**
```
public override bool Add()
```

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the RadioButton field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFRadioButtonField class with PDFDocument object
            PDFRadioButtonField radioButtonField = new

                  PDFRadioButtonField(document);
            // set the PDFRadioButtonField field Metadata parameters
            radioButtonField.Metadata.StartingX = 100;
            radioButtonField.Metadata.StartingY = 100;
            radioButtonField.Metadata.Height = 150;
            radioButtonField.Metadata.Width = 300;
            radioButtonField.Metadata.PageNumber = 1;
            radioButtonField.Metadata.Tooltip = "RadioButton1";
            radioButtonField.Metadata.FieldName = "RadioButton1";
            radioButtonField.Metadata.Visibility = VisibilityOptions.Visible;
            radioButtonField.Metadata.Orientation =
                  OrientationOptions.ZeroDegrees;
            radioButtonField.Metadata.ReadOnly = false;
            radioButtonField.Metadata.Required = false;
            radioButtonField.Metadata.Locked = false;
            // set the PDFRadioButtonField field style parameters
            radioButtonField.Style.BorderColor = System.Drawing.Color.Empty;
            radioButtonField.Style.BorderWidth = BorderWidthOptions.Thin;
            radioButtonField.Style.FillColor =  System.Drawing.Color.Empty;
            radioButtonField.Style.FontColor =  System.Drawing.Color.Empty;
            radioButtonField.Style.FontFace = FontFaces.Helvetica;
            radioButtonField.Style.FontSize = 0;
            radioButtonField.Style.BorderStyle = BorderStyles.Solid;
            radioButtonField.CheckedByDefault = false;
            radioButtonField.CheckStyle = CheckStyles.Check;
            radioButtonField.ExportValue = "Yes";
            radioButtonField.ExportIndex = 1;
            radioButtonField.Unision = false;
            // add the RadioButton field
            radioButtonField.Add();
            ...
        }
    }
}
```

### 7.7.3.2 The Delete method

**Method Description**
Deletes a RadioButton field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Delete()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the RadioButton field was successfully deleted.

**Usage**
```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFRadioButtonField class with PDFDocument object
                PDFRadioButttonField  radioButtonField = new

                PDFRadioButtonField(document);
            // delete the radio button field
            radioButtonField.Metadata.FieldName = "RadioButton1";
            bool result = radioButtonField.Delete();
            if (result)
                Console.WriteLine("RadioButton field deleted successfully.");
            else
                Console.WriteLine("Error: {0}", radioButtonField.ErrorMessage);
       }
    }
}
```

### 7.7.3.3 The SetState method

**Method Description**
Sets the state of the specified RadioButton field.

**Syntax**
public bool SetState(string exportValue)

**Parameters**
*exportValue*
Type: string
The export value of the RadioButton field.

**Return Value**
A boolean value indicating if the state was successfully set.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            FillField();
        }

        public static void FillField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFRadioButton class with with PDFDocument object
            PDFRadioButtonField  radioButtonField = new
                                           PDFRadioButtonField(document);
            // set the radio button field state (check)
            radioButtonField.Metadata.FieldName = "RadioButton1";
            string exportValue = radioButtonField.GetValue();
            bool result = radioButtonField.SetState(exportValue);
            if (result)
                Console.WriteLine("Radio button state set successfully.");
            else
                Console.WriteLine("Error: {0}", radioButtonField.ErrorMessage);
        }
    }
}
```

#### 7.7.3.4 The GetStates method

**Method Description**
Gets the states of the specified RadioButton field.

**Syntax**
public List<string> GetStates()

**Parameters**
*None.*

**Return Value**
The export values of the RadioButton.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetStates();
        }

        public static void GetStates()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFRadioButtonField class with PDFDocument object
            PDFRadioButtonField radioButtonField = new

                  PDFRadioButtonField(document);
            // get the export values for the RadioButton field
            radioButtonField.Metadata.FieldName = "RadioButton1";
            List<string> exportValues = radioButtonField.GetStates();
            // loop through the export values
            ...

        }
    }
}
```

### 7.7.3.5 The GetValue method

**Method Description**
Gets the export value of RadioButton field.

**Syntax**
public string GetValue()

**Parameters**
*None.*

**Return Value**
The export value of RadioButton field.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetValue();
        }

        public static void GetValue()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFRadioButtonField class with the PDFDocument
            object
            PDFRadioButtonField  radioButtonField = new
                                           PDFRadioButtonField(document);
            radioButtonField.Metadata.FieldName = "RadioButton1";
            // get the export value of the RadioButton field
            string exportValue = radioButtonField.GetValue();
            if (exportValue != null)
                Console.WriteLine("Export Value: {0}", exportValue);
            else
                Console.WriteLine("Error: {0}", radioButtonField.ErrorMessage);
        }
    }
}
```

## 7.8 The PDFSignField class

The main class to add a signature field to a PDF document. This class can be used to support signing of PDF documents without predefined digital signature fields. Use this class to add a signature field to the PDF document before sending the document for signing. This class inherits the base PDFFormField class.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.FormFields**

**Syntax**
public class PDFSignField : PDFFormField

### 7.8.1 Constructors

**Syntax**
public PDFSignField(PDFDocument pdfDocument) : base()

**Parameters**
*pdfDocument*
Type: PDFDocument object
The PDFDocument object.

### 7.8.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | ReadOnlyFormFields | public IReadOnlyFormField ReadOnlyFormFields{ get; } | The read-only form field parameters. Exposed through IReadOnlyFormField interface. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

## 7.8.3 Methods

### 7.8.3.1 The Load method

**Method Description**
Loads the PDFSignField object.

**Syntax**
public bool Load(string fieldName)

**Parameters**
*fieldName*
Type: string
The name that identifies the signature field.

**Return Value**
A boolean value indicating if the PDFSignField object was successfully loaded.
**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            LoadField();
        }

        public static void LoadField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSignField class with PDFDocument object
            PDFSignField signField = new PDFSignField(document);
            signField.Load("Signature1");
            // read the PDFSignField Metadata parameters
            ...
            // read the PDFSignField style parameters
            ...
        }
    }
}
```

### 7.8.3.2 The Add method

**Method Description**
Adds a signature field to a PDF document. Inherited and overridden from base class PDFFormField.

**Syntax**
```
public override bool Add()
```

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the signature field was successfully added.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddField();
        }

        public static void AddField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSignField class with PDFDocument object
            PDFSignField signField = new PDFSignField(document);
            // set the PDFSignField Metadata parameters
            signField.Metadata.StartingX = 100;
            signField.Metadata.StartingY = 100;
            signField.Metadata.Height = 150;
            signField.Metadata.Width = 300;
            signField.Metadata.PageNumber = 1;
            signField.Metadata.Tooltip  = "Signature1";
            signField.Metadata.FieldName  = "Signature1";
            signField.Metadata.Visibility = VisibilityOptions.Visible;
            signField.Metadata.Orientation = OrientationOptions.ZeroDegrees;
            signField.Metadata.ReadOnly = false;
            signField.Metadata.Required = false;
            signField.Metadata.Locked = false;
            // set the PDFSignField style parameters
            signField.Style.BorderColor  = System.Drawing.Color.Empty;
            signField.Style.BorderWidth = BorderWidthOptions.Thin;
            signField.Style.FillColor = System.Drawing.Color.Empty;
            signField.Style.FontColor = System.Drawing.Color.Empty;
            signField.Style.FontFace = FontFaces.Helvetica;
            signField.Style.BorderStyle = BorderStyles.Solid;
            signField.ReadOnlyFormFields.FieldListType = FieldListTypes.None;
            // add the signature field
            bool result = signField.Add();;
            if (result)
                Console.WriteLine("Signature field added successfully.");
            else
                Console.WriteLine("Error: {0}", signField.ErrorMessage);

        }
    }
}
```

### 7.8.3.3 The Delete method

**Method Description**
Deletes a signature field from a PDF Document. Inherited and overridden from base class PDFFormField.

**Syntax**
public override bool Delete()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the signature field was successfully deleted.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.FormFields;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            DeleteField();
        }

        public static void DeleteField()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with the input document
            PDFDocument document =  new PDFDocument(inputDocument);
            // initialize the PDFSignField class with PDFDocument object
            PDFSignField  signField= new PDFSignField(document);
            // delete the signature field
            signField.Metadata.FieldName = "Signature1";
            bool result = signField.Delete();
            if (result)
                Console.WriteLine("Signature field deleted successfully.");
            else
                Console.WriteLine("Error: {0}", signField.ErrorMessage);
        }
    }
}
```

## 7.9 The PDFSignDictionary class

A class to create a custom dictionary that can be added to the PDF document before signing.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries**
**Syntax**
public class PDFSignDictionary

### *7.9.1 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | Name | public string Name { get; set; } | The name of the dictionary. |
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### *7.9.2 Methods*

#### *7.9.2.1 The AddItem method*

**Method Description**
Adds an item to the dictionary in the form of name and value.

**Syntax**
public bool AddItem(string name, string value)

**Parameters**
*name*
Type: string
The name that identifies the item to be added.
*value*
Type: string
The value to be set against the name.

**Return Value**
A boolean value indicating if the name/value was successfully added.

**Usage**

The following sample shows how to add a dictionary item to a PDFSignDictionary instance. One can add any number of items to the PDFSignDictionary instance. Item names in a single PDFSignDictionary instance should have unique names.

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddItemToDictionary();
        }
        public static void AddItemToDictionary()
        {
            // create a custom dictionary to be added to the signature
            PDFSignDictionary dictionary = new PDFSignDictionary();
            // add an item to the PDFSignDictionary object
            dictionary.Name = "CustomDictionary";
            bool result = dictionary.AddItem("CustomItem1", "CustomValue1");
            if (result)
                Console.WriteLine("Item added.");
            else
                Console.WriteLine("Error: {0}", dictionary.ErrorMessage);
        }
    }
}
```

### *7.9.2.2 The RemoveItem method*

**Method Description**

Removes an item from the dictionary based on the name specified.

**Syntax**

public bool RemoveItem(string name)

**Parameters**

*name*

Type: string

The name that identifies the item to be removed.

**Return Value**

A boolean value indicating if the item was successfully removed.

**Usage**

The following sample shows how to remove a dictionary item from a PDFSignDictionary instance.

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            RemoveItemFromDictionary();
        }

        public static void RemoveItemFromDictionary ()
        {
            // create a custom dictionary to be added to the signature
            PDFSignDictionary dictionary = new PDFSignDictionary();
            // remove an item from the PDFSignDictionary object
            dictionary.Name = "CustomDictionary";
            bool result = dictionary.RemoveItem("CustomItem1");
            if (result)
                Console.WriteLine("Item removed.");
            else
                Console.WriteLine("Error: {0}", dictionary.ErrorMessage);
        }
    }
}
```

### 7.9.2.3 The GetItem method

**Method Description**

Gets the value for the corresponding name in the custom dictionary.

**Syntax**

public string GetItem(string name)

**Parameters**

*name*

Type: string

The name that identifies the item whose value is to be retrieved.

**Return Value**

The value for the corresponding name in the custom dictionary.

**Usage**

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetItemValue();
        }

        public static void GetItemValue()
        {
            // create a custom dictionary to be added to the signature
            PDFSignDictionary dictionary = new PDFSignDictionary();
            string itemValue = dictionary.GetItem("CustomItem1");
            // get an item from the PDFSignDictionary object
            dictionary.Name = "CustomDictionary";
            if (itemValue != null)
                Console.WriteLine("Item Value: {0}", itemValue);
            else
                Console.WriteLine("Error: {0}", dictionary.ErrorMessage);
        }
    }
}
```

## 7.10 The PDFSign class

The main class for signing a PDF document. This class can be used for a single pass as well as two pass signing. Refer to Appendix A for detailed information on single pass and two pass signing.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFSigner**

**Syntax**
public class PDFSign

### *7.10.1 Constructors*

**Syntax**
public PDFSign (PDFDocument pdfDocument)

**Parameters**
*pdfDocument*
Type: PDFDocument object.
The PDFDocument object.

### 7.10.2 Properties

| | Name | Syntax | Description |
|---|---|---|---|
| | SignatureAction | public SignatureActions SignatureAction { get; set; } | The signature action defined by the SignatureActions enumeration. |
| | Appearance | public IAppearanceInfo Appearance { get; } | The signature field appearance. Exposed through IAppearanceInfo interface. |
| | Store | public IStoreInfo Store{ get; } | The store information that is to be used for signing. Exposed through IStoreInfo interface. Required for single pass signing. |
| | Certificate | public ICertificateInfo Certificate { get; } | The certificate information that is used for signing. Exposed through ICertificateInfo interface. Required for two pass signing. |
| | Signature | public ISignatureInfo Signature { get; } | The signature information. Exposed through ISignatureInfo interface. |
| | ContentHash | public byte[] ContentHash { get; } | The hash of contents saved in the signature dictionary. |
| | ErrorMessage | public string ErrorMessage{ get; } | Gets the message that describes the last error. |

### 7.10.3 Methods

#### *7.10.3.1 The AddCustomDictionary method*

**Method Description**
Adds a custom dictionary to the signature.

**Syntax**
public bool AddCustomDictionary(PDFSignDictionary customDictionary)

**Parameters**
*customDictionary*
Type: PDFSignDictionary object
The PDFSignDictionary object.

**Return Value**
A boolean value indicating if the custom dictionary was successfully added.

**Usage**
The following sample shows how to add the custom dictionary to the signature. The dictionary should be added before calling the method to sign the document.

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;
namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            AddDictionary();
        }
        public static void AddDictionary()
        {
            // create a custom dictionary to be added to the signature
            ...
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // add the custom dictionary to the signature
            bool result = sign.AddCustomDictionary(dictionary);
        }
    }
}
```

### *7.10.3.2 The RemoveCustomDictionary method*

**Method Description**
Removes a custom dictionary from the signature.

**Syntax**
public bool RemoveCustomDictionary(string dictionaryName)

**Parameters**
*dictionaryName*
Type: string
The custom dictionary name.

**Return Value**
A boolean value indicating if the custom dictionary was successfully removed.

**Usage**

The following sample shows how to remove the custom dictionary from a signature. The dictionary with the specified name should exist in the PDFSign instance. The dictionary should be removed before calling the method to sign the document.

```
using System;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            RemoveDictionary();
        }

        public static void RemoveDictionary()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // remove the custom dictionary from the signature
            bool result = sign.RemoveCustomDictionary("CustomDictionary");
            if (result)
                Console.WriteLine("Custom dictionary removed.");
            else
                Console.WriteLine("Error: {0}", sign.ErrorMessage);
        }
    }
}
```

### *7.10.3.3 The Sign method*

**Method Description**
Signs a PDF document in a single pass.

**Syntax**
public byte[] Sign(string fieldName)

**Parameters**
*fieldName*
Type: string
The name of the signature field to be signed.

**Return Value**
The signed PDF document bytes. Null in case of an error.

**Usage**

The following sample shows how to sign a signature field in a PDF document in a single pass.

```
using System;
using System.IO;
using System.Drawing;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Sign();
        }

        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            ...
            // define the appearance of the signature
            ...
            // manage dictionaries
            ...
            // set the store data
            ...
            // signature action can be "Sign" or "Initials"
            sign.SignatureAction = SignatureActions.Sign;
            // sign the document
            byte[] outputDocument = sign.Sign("Signature1");
            if (outputDocument != null)
            {
                // write the out document to the file system
                using (FileStream fs = File.Create(@"C:\Sample-Output.pdf"))
                {
                    BinaryWriter bw = new BinaryWriter(fs);
                    bw.Write(outputDocument);
                    bw.Close();
                    fs.Close();
                }
            }
            else
                Console.WriteLine("Error: {0}", sign.ErrorMessage);
        }
    }
}
```

### 7.10.3.4 The PreSign method

**Method Description**
Pre-signs the PDF document by inserting a dummy signature in the document. First part of the two passes signing. The signing should be completed by making a call to post-sign after signing the content hash (can be accessed using the property ContentHash of the PDFSign Class).

**Syntax**
```
public byte[] PreSign(string fieldName)
```

**Parameters**
*fieldName*
Type: string
The name of the signature field to be signed.

**Return Value**
The signed (dummy signature) PDF document bytes. Null in case of an error.

### 7.10.3.5 The PostSign method

**Method Description**
Post-signs (Applies) the signature by replacing the dummy signature inserted during Pre-sign with actual signature.

**Syntax**
```
public byte[] PostSign(byte[] hash, byte[] signature, string fieldName)
```

**Parameters**
*hash*
Type: byte array
The hash that was signed.
*signature*
Type: byte array
Externally computed PKCS7 signature.
*fieldName*
Type: string
The name of the signature field that was pre-signed.

**Return Value**
The signed PDF document bytes. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using System.Drawing;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFSigner.SignatureAppearance;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Sign();
        }

        public static void Sign()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the signature information
            ...
            // define the appearance of the signature
            ...
            // manage dictionaries
            ...
            // set the certificate
            sign.Certificate.CertificateData = File.ReadAllBytes(@"C:\Sign.cer");
            // signature action can be "Sign" or "Initials"
            sign.SignatureAction = SignatureActions.Sign;
            byte[] dummy = sign.PreSign("Signature1");
            if (dummy == null)
            {
                  Console.WriteLine("Error: {0}", sign.ErrorMessage);
                  return;
            }
```

Continued...

```
            // get the hash and sign (external signature)
            byte[] hash = sign.ContentHash;
            byte[] sig = GetRSASignature(hash);
            sign = null;
            sign = new PDFSign(dummy);
            // set the certificate
            sign.Certificate.CertificateData = File.ReadAllBytes(@"C:\Sign.cer");
            byte[] outputDocument = sign.PostSign(hash, sig, "Signature1");
            if (outputDocument != null)
            {
                // write the out document to the file system
                using (FileStream fs = File.Create(@"C:\Sample-Output.pdf"))
                {
                    BinaryWriter bw = new BinaryWriter(fs);
                    bw.Write(outputDocument);
                    bw.Close();
                    fs.Close();
                }
            }
            else
                Console.WriteLine("Error: {0}", sign.ErrorMessage);
        }

        private static byte[] GetRSASignature(byte[] hashToSign)
        {
            string pfxFilePath = @"C:\Sign.pfx";
            string password = "password";
            X509Certificate2 cert = new X509Certificate2(pfxFilePath, password,
                                                X509KeyStorageFlags.UserKeySet);
            // Use Case: Certificate uses PROV_RSA_FULL
            RSACryptoServiceProvider rsaCSP =
                        (RSACryptoServiceProvider)cert.PrivateKey;
            CspParameters cspParameters = new CspParameters();
            cspParameters.KeyContainerName =
                        rsaCSP.CspKeyContainerInfo.KeyContainerName;
            cspParameters.KeyNumber = rsaCSP.CspKeyContainerInfo.KeyNumber ==
                                                      KeyNumber.Exchange ? 1 : 2;
            // As of .NET 3.5 SP1 the CSP instance works with a provider of type
            // PROV_RSA_AES
            RSACryptoServiceProvider rsa = new
                                    RSACryptoServiceProvider(cspParameters);
            rsa.PersistKeyInCsp = false;
            byte[] signBytes = rsa.SignHash(hashToSign,
                                          CryptoConfig.MapNameToOID("SHA256"));
            rsa.Clear();
            return signBytes;
        }
    }
}
```

### 7.10.3.6 The Certify method

**Method Description**
Certifies a PDF document.

**Syntax**
public byte[] Certify()

**Parameters**
*None*

**Return Value**
The certified PDF document bytes. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            Certify();
        }
        public static void Certify()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFSign class with the PDFDocument object
            PDFSign sign = new PDFSign(document);
            // set the store data
            ...
            // Set the signature Information
            ...
            certifyDocumentBytes = sign.Certify();
            if (certifyDocumentBytes != null)
            {
                // write the out document to the file system
                using (FileStream fs = File.Create(@"C:\Sample-Output.pdf"))
                {
                    BinaryWriter bw = new BinaryWriter(fs);
                    bw.Write(certifyDocumentBytes );
                    bw.Close();
                    fs.Close();
                }
            }
            else
                  Console.WriteLine("Error: {0}", sign.ErrorMessage);
        }
    }
}
```

## 7.11 The PDFVerify class

This is the main class for verifying a PDF document. This class can be used to verify a signature and to verify a certificate. It also contains other methods that expose verification details.

**Namespace**
**Topaz.MultiPlatformSDK.Embed.PDFVerfier**

**Syntax**
public class PDFVerify

### *7.11.1 Constructors*

**Syntax**
public PDFVerify(PDFDocument pdfDocument)

**Parameters**
*pdfDocument*
Type: PDFDocument object.
The PDFDocument object.

### *7.11.2 Properties*

| | Name | Syntax | Description |
|---|---|---|---|
| | ErrorMessage | public string ErrorMessage { get; } | Gets the message that describes the last error. |

### *7.11.3 Methods*

#### *7.11.3.1 The IsDocumentValid method*

**Method Description**
Checks if the document initialized is a valid PDF document.

**Syntax**
public bool IsDocumentValid()

**Parameters**
*None.*

**Return Value**
A boolean value indicating if the document initialized is a valid PDF document.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            CheckDocumentValidity();
        }

        public static void CheckDocumentValidity()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            bool isValid = verify.IsDocumentValid();
            if (isValid)
               Console.WriteLine("Document is valid.");
            else
               Console.WriteLine("Document is not valid.");
        }
    }
}
```

### *7.11.3.2 The GetUnsignedSignatureFieldNames method*

**Method Description**

Gets the list of unsigned signature field names in the PDF document. Deprecated – Use GetFieldList() method (Section 7.1.3.1) instead.

**Syntax**

public List<string> GetUnsignedSignatureFieldNames()

**Parameters**

*None.*

**Return Value**

The list of unsigned signature field names. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using System.Collections.Generic;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetUnsignedSignatureFieldNames();
        }

        public static void GetUnsignedSignatureFieldNames()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            List<string> list = verify.GetUnsignedSignatureFieldNames();
            if (list != null)
            {
               foreach (string fieldName in list)
               {
                  Console.WriteLine("Unsigned Field: {0}", fieldName);
               }
            }
            else
               Console.WriteLine("Error: {0}", verify.ErrorMessage);

        }
    }
}
```

### *7.11.3.3 The GetSignedSignatureFieldNames method*

**Method Description**

Gets the list of signed signature field names in the PDF document. Deprecated – Use GetFieldList() method (Section 7.1.3.1) instead.

**Syntax**

public List<string> GetSignedSignatureFieldNames()

**Parameters**

*None.*

**Return Value**

The list of signed signature field names. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using System.Collections.Generic;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetUnsignedSignatureFieldNames();
        }

        public static void GetUnsignedSignatureFieldNames()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            List<string> list = verify.GetSignedSignatureFieldNames();
            if (list != null)
            {
               foreach (string fieldName in list)
               {
                  Console.WriteLine("Unsigned Field: {0}", fieldName);
               }
            }
            else
               Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### 7.11.3.4 The VerifySignature method

**Method Description**
Verifies the signature in a PDF document. The verification details are exposed through the *ISignatureVerificationDetails* interface.

**Syntax**
public ISignatureVerificationDetails VerifySignature(string fieldName)

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.

**Return Value**
The signature verification details exposed through the *ISignatureVerificationDetails* interface.
Null in case of an error.

**Usage**

```
using System;
using System.IO;
using System.Collections.Generic;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            VerifySignature();
        }

        public static void VerifySignature()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            ISignatureVerificationDetails details =
                                              verify.VerifySignature("Signature1");
            if (details != null)
            {
               string result = null;
               switch (details.Result)
               {
                 case SignatureVerificationResults.SignatureInvalid:
                    result = "Signature invalid";
                    break;
                 case SignatureVerificationResults.SignatureValidDocumentChanged:
                    result = "Signature valid but document changed";
                    break;
                                case
               SignatureVerificationResults.SignatureValidDocumentNotChanged:
                    result = "Signature valid and document not changed";
                    break;
               }
               Console.WriteLine("Result: {0}", result);
               Console.WriteLine("Message: {0}", details.Message);
               Console.WriteLine("Revision: {0}", details.Revision.ToString());
               Console.WriteLine("TotalRevisions: {0}",
                                          details.TotalRevisions.ToString());
             }
             else
                Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### *7.11.3.5 The VerifyCertificate method*

**Method Description**
Verifies the certificate that was used for signing. The verification details are exposed through the *ICertificateVerificationDetails* interface.

**Syntax**
public ICertificateVerificationDetails VerifyCertificate(string fieldName)

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.

**Return Value**
The certificate verification details exposed through the *ICertificateVerificationDetails* interface.
Null in case of an error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            VerifyCertificate();
        }

         public static void VerifyCertificate()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
              // initialize the PDFDocument class with input document
              PDFDocument document = new PDFDocument(inputDocument);
              // initialize the PDFVerify class with the PDFDocument object
              PDFVerify verify = new PDFVerify(document);
            ICertificateVerificationDetails certDetails =
                                                 verify.VerifyCertificate("Signature1");
              if (certDetails != null)
            {
               string result = null;
               switch (certDetails.Result)
               {
                  case CertificateVerificationResults.VerificationPassed:
                     result = "Certificate Verification Passed";
                     break;
                  case CertificateVerificationResults.VerificationFailed:
                     result = "Certificate Verification Failed";
                     break;
                 }
               Console.WriteLine("Result: {0}", result);
               Console.WriteLine("Message: {0}", certDetails.Message);
              }
            else
               Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### *7.11.3.6 The GetSignatureInfo method*

**Method Description**
Extracts the signature information from the signature dictionary. The signature information are exposed through ISignatureInfo interface.

**Syntax**
public ISignatureInfo GetSignatureInfo(string fieldName)

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.

**Return Value**
The signature details are exposed through *ISignatureInfo* interface. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetSignatureInfo();
        }

        public static void GetSignatureInfo()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
             // initialize the PDFDocument class with input document
             PDFDocument document = new PDFDocument(inputDocument);
             // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
             ISignatureInfo signInfo = verify.GetSignatureInfo("Signature1");
            if (signInfo != null)
            {
               Console.WriteLine("Filter: {0}", signInfo.Filter);
               Console.WriteLine("SubFilter: {0}", signInfo.SubFilter);
               Console.WriteLine("FieldType: {0}", signInfo.FieldType);
               Console.WriteLine("SignDateTime: {0}",
                                           signInfo.SignDateTime.ToString());
               Console.WriteLine("SignerName: {0}", signInfo.SignerName);
               Console.WriteLine("Reason: {0}", signInfo.Reason);
               Console.WriteLine("Location: {0}", signInfo.Location);
               Console.WriteLine("Contents: {0}", signInfo.Contents);
            }
            else
               Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### 7.11.3.7 The GetCertificateDetails method

**Method Description**
Gets the details of the certificate used for signing. The certificate details are exposed through *ICertificateDetails* interface.

**Syntax**
```
public ICertificateDetails GetCertificateDetails(string fieldName)
```

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.

**Return Value**
The certificate details exposed through *ICertificateDetails* interface. Null in case of an error.

### Usage

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFSigner;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetCertificateDetails();
        }

        public static void GetCertificateDetails()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
               // initialize the PDFDocument class with input document
               PDFDocument document = new PDFDocument(inputDocument);
               // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
               ICertificateDetails certInfo =
                                              verify.GetCertificateDetails("Signature1");
            if (certInfo != null)
            {
                Console.WriteLine("IssuedTo: {0}", certInfo.IssuedTo);
                Console.WriteLine("Issuer: {0}", certInfo.Issuer);
                Console.WriteLine("Thumbprint: {0}", certInfo.Thumbprint);
                Console.WriteLine("BeginDate: {0}",
                                                 certInfo.BeginDate.ToString());
                   Console.WriteLine("EndDate: {0}", certInfo.EndDate.ToString());
                Console.WriteLine("SerialNumber: {0}", certInfo.SerialNumber);
                Console.WriteLine("VersionNumber: {0}",
                                                certInfo.VersionNumber.ToString());
                Console.WriteLine("PublicKey: {0}", certInfo.PublicKey);
                Console.WriteLine("PublicKeyAlgorithm: {0}",
                                                certInfo.PublicKeyAlgorithm.ToUpper());
                Console.WriteLine("SignatureAlgorithm: {0}",
                                                certInfo.SignatureAlgorithm.ToUpper());
            }
            else
                Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### 7.11.3.8 The GetCustomDictionary method

**Method Description**
Gets the custom dictionary and the associated child dictionaries.

**Syntax**
public PDFSignDictionary GetCustomDictionary(string fieldName, string dictionaryName)

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.
*dictionaryName*
Type: string
The name of the custom dictionary.

**Return Value**
The custom dictionary and the associated child dictionaries.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFSignatureDictionaries;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetCustomDictionary();
        }

        public static void GetCustomDictionary()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            PDFSignDictionary customDict =
                        verify.GetCustomDictionary("Signature1",
                  "CustomDictionary");
            ...

        }
    }
}
```

### *7.11.3.9 The GetRevisionFieldName method*

**Method Description**
Gets the field that was signed for the given revision.

**Syntax**
public string GetRevisionFieldName(int revisionNo)

**Parameters**
*revisionNo*
Type: integer
The revision number.

**Return Value**
The field that was signed for the given revision. Null in case of an error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetRevisionFieldName();
        }

        public static void GetRevisionFieldName()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            string revisionFieldName = verify.GetRevisionFieldName(1);
            if (!String.IsNullOrEmpty(revisionFieldName))
                Console.WriteLine("Revision Field Name: {0}", revisionFieldName);
            else
                Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

### 7.11.3.10 The GetDocumentRevision method

**Method Description**
Gets the PDF bytes corresponding to a particular revision of the document.

**Syntax**
public byte[] GetDocumentRevision(int revisionNo)

**Parameters**
*revisionNo*
Type: integer
The revision number.

**Return Value**
The PDF bytes corresponding to a particular revision of the document. Null in case of error.

**Usage**

```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetDocumentRevision();
        }

        public static void GetDocumentRevision()
        {
            byte[] inputDocument = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            byte[] pdfBytes = verify.GetDocumentRevision(1);
        }
    }
}
```

### 7.11.3.11 The GetSignerName method

**Method Description**
Gets the name stored in the signature dictionary.

**Syntax**
public string GetSignerName(string fieldName)

**Parameters**
*fieldName*
Type: string
The name of the signature field used for signing.

**Return Value**
The name. Null in case of an error.

**Usage**

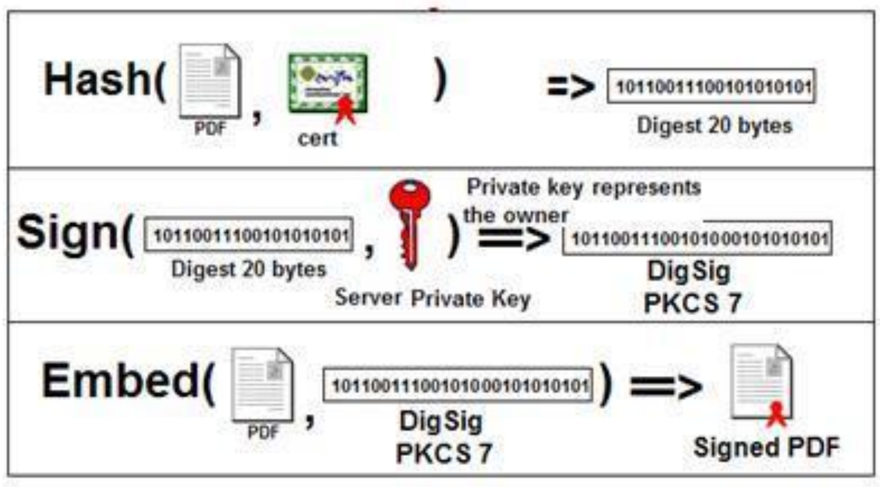
```
using System;
using System.IO;
using Topaz.MultiPlatformSDK.Embed.PDFVerfier;
using Topaz.MultiPlatformSDK.Embed.PDFDocuments;

namespace Topaz.MultiPlatformSDK.Samples
{
    public class Example
    {
        public static void Main(string[] args)
        {
            GetSignerName();
        }

        public static void GetSignerName()
        {
            byte[] documentBytes = File.ReadAllBytes(@"C:\Sample.pdf");
            // initialize the PDFDocument class with input document
            PDFDocument document = new PDFDocument(inputDocument);
            // initialize the PDFVerify class with the PDFDocument object
            PDFVerify verify = new PDFVerify(document);
            string signerName = verify.GetSignerName("Signature1");
            if (!String.IsNullOrEmpty(signerName))
               Console.WriteLine("Signer Name: {0}", signerName);
            else
               Console.WriteLine("Error: {0}", verify.ErrorMessage);
        }
    }
}
```

# 8.0 – Appendix A

This section explains the single pass and two pass signing process.

### 8.1 Single Pass Signing

Single pass signing is applicable for scenarios where the PDF document and the signing certificate and private key are at the same location, and signing happens in a single step as all the required inputs are available. The following diagram shows the single pass signing process.

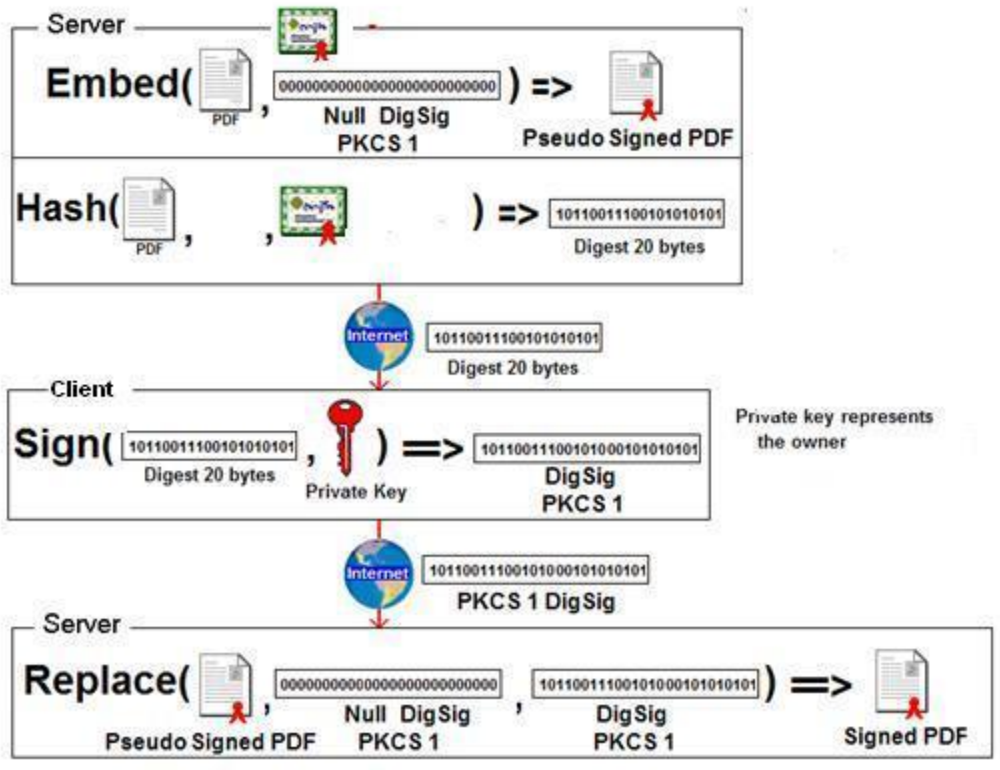

Hash( PDF , cert ) => 10110011100101010101
Digest 20 bytes
Sign( 10110011100101010101 , ) => 1011001110010100010101010101
Digest 20 bytes
Server Private Key
Private key represents the owner
Dig Sig PKCS 7
Embed( PDF , 1011001110010100010101010101 ) =>
Dig Sig PKCS 7
Signed PDF

Single Pass Signing

## 8.2 Two Pass Signing

In two pass signing, the signing is a fragmented and distributed system for applying verifiable signatures. This is useful for scenarios where the private key to be used for singing is remote to the PDF document to be signed. The signing happens in two phases, namely pre-sign and post-sign. The pre-sign phase signs the signature in the document using a dummy signature and computes the hash of the PDF document. The host application has to sign the hash using the private key of the signer and then use post sign to replace the dummy signature with the actual signature.

This is useful for scenarios where the private key is residing on the client desktop or mobile device and the PDF document and singing happens on a web server. The following diagram explains the two pass signing process.


Server
Embed( PDF , 00000000000000000000000000 Null DigSig PKCS 1 ) => Pseudo Signed PDF
Hash( PDF , , ) => 10110011100101010101 Digest 20 bytes
Internet
10110011100101010101 Digest 20 bytes
Client
Sign( 10110011100101010101 Digest 20 bytes , Private Key ) => 101100111001010001010101 DigSig PKCS 1
Private key represents the owner
Internet
101100111001010001010101 PKCS 1 DigSig
Server
Replace( Pseudo Signed PDF , 00000000000000000000000000 Null DigSig PKCS 1 , 101100111001010001010101 DigSig PKCS 1 ) => Signed PDF

Two Pass Signing