eSign Emcee Proxy Handler Implementation Guide

eSign Emcee is a trademark of eSign Emcee
All other trademarks are the property of their respective owners.

www.esignemcee.com
Copyright © 2016 eSign Emcee

Table Of Contents

1. Introduction 3

2. Overview 3

3. eSign Emcee Web Services Methods 4

4. “Proxy” Implementation 4

4.1 Reading the input document. 4
4.1.1 Merging FDF with PDF 6

4.2 Create XML formatted additional data 9
4.2.1 The XML structure 14
4.2.1.1 Signing Ceremony <SigningCeremony /> 14
4.2.1.2 Index File Creation <IndexMapper /> 16
4.2.1.3 Signature Prefixes <SignaturePrefixes /> 16
4.2.2. XML Sample 17

4.3 Configure the Web.config 19

4.4 Call the eSign Emcee Web Service 20

4.5 Launch the eSign Emcee “Presenter” 21

 Copyright © 2016 eSign Emcee

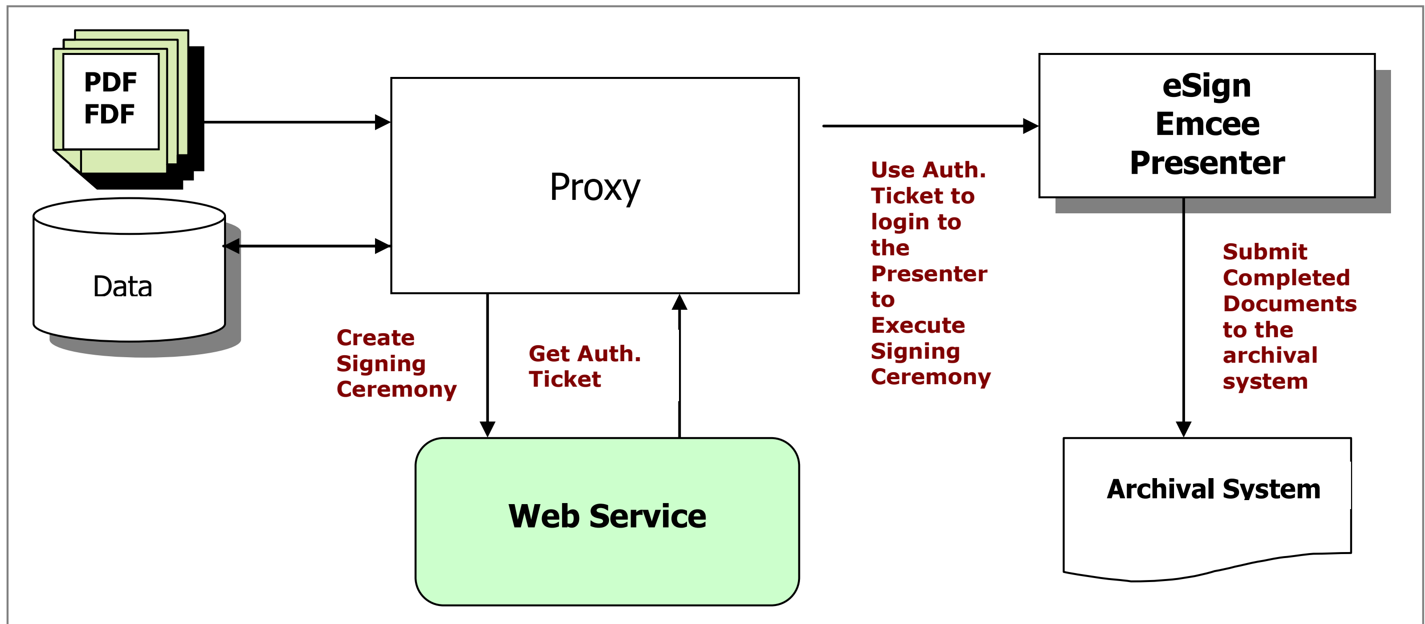
1. Introduction

eSign Emcee creates and executes signing ceremonies. To create a signing ceremony eSign Emcee requires a minimum set of information. This information is traditionally collected by the eSign Emcee Designer module. The designer module creates a 'template' by collecting the information entered by the end user.

In certain instances however it is desirable to submit single PDF forms for signature execution using the Adobe Acrobat PDF/FDF document submission standard. By leveraging eSign Emcee Web Services, the ceremony creation process can be automated. To enable this, eSign Emcee Web Service exposes a number of methods to enable the signing ceremony creation. Some of the information required by eSign Emcee is contained in the PDF/FDF data itself. However, eSign Emcee requires some additional data to be able to create a signing ceremony. This is where the implementation of an indirect or "Proxy Handler" application becomes useful.

2. Overview

The "Proxy Handler" application receives the PDF/FDF data and creates a pre-defined eSign Emcee XML containing the additional data as required by the eSign Emcee Web Service to create a signing ceremony. The source of data for the XML can be anything. It can be a database, another XML, the PDF itself or a combination of one or all of the given sources. The primary job of the proxy is to receive the PDF/FDF data, merge the source template PDF with the received form data and then create the required ceremony XML with the additional data for the eSign Emcee Web Service to consume.



 Copyright © 2016 eSign Emcee

3. eSign Emcee Web Services Methods

The eSign Web Services module exposes a rich set of methods that enable end users to create their own applications to manage eSign Emcee tasks. These tasks include user management, signing ceremony creation and authentication into the eSign Emcee system.

The “Proxy” handler uses a small subset of these methods to help create and execute ceremonies in eSign Emcee. Specifically, the proxy handler uses the following methods to achieve its objective:

- Connect()
- CreateSigningCeremonyFromDocument()
- Disconnect()

This document will restrict itself to “Proxy” implementation guidelines. For detailed documentation of these methods as well as the complete set of methods please refer the ***eSign Emcee Web Services Integration Document***.

4. “Proxy” Implementation

This section gives a brief overview of how to implement a PDF/FDF “Proxy” handler. It is assumed that a correct reference of the eSign Emcee Web Service has been added to the project. Since proxy is a customizable application please note that the document explains the various steps involved with code snippets.

4.1 Reading the input document.

The first step in processing the PDF/FDF data stream is to read the input document and convert it to a base64 string before calling the web service. The following code snippet illustrates how to read the input document. The code needs to be added to the “Page_Load” event in the page.

```
          byte[] pdfData = null;
bool error = false;
string errorMessage = null;
Stream req = null;
try
{
   // Read the stream and check if a valid request was made
   req = Request.InputStream;
   string domainName = ConfigurationManager.AppSettings["DomainName"];
   if (!domainName.Equals(Request.Url.Host,
      StringComparison.InvariantCultureIgnoreCase))
   {
      Trace.Write("URL authentication failed. URL does not match with
               configured url: "
         + domainName + " != " + Request.Url.Host);
      throw new Exception("URL authentication failed. Request not from
                    trusted URL domain.");
   }
   if (req.Length == 0)
   {
      throw new Exception("Input request was null. Invalid input
                     stream.");
   }
```


Copyright © 2016 eSign Emcee

```
    byte[] inputData = new byte[req.Length];
    req.Position = 0;
    req.Read(inputData, 0, inputData.Length);
    Trace.Write("Read input data.");
    if (Request.ContentType == "application/vnd.fdf")
    {
        string err = null;
        Trace.Write("Application type: application/vnd.fdf");
        pdfData = GetMergedPdfBytes(inputData, out err);
        Trace.Write("Got Pdf bytes");
        if (pdfData == null)
            throw new Exception(err);
    }
    else if (Request.ContentType == "application/pdf")
    {
        Trace.Write("Application type: application/pdf");
        pdfData = inputData;
    }
    else
    {
        throw new Exception("Input Request Content Type is not a valid
                    application.");
    }
}
catch (Exception ex)
{
    errorMessage = "Error Details: " + ex.Message;
    Trace.Warn(errorMessage);
    lblErrorHead.Text = "An error occured while executing the request.";
    lblErrorDetails.Text = errorMessage;
    error = true;
}
finally
{
    // close the request stream
    if (req != null)
    {
        req.Close();
    }
}
```


Copyright © 2016 eSign Emcee

4.1.1 Merging FDF with PDF

The following code snippet shows how to compute the PDF security password, merge the FDF into PDF and manipulate the PDF [show or hide the fields, add transparency and flatten some of the fields] and return the same.

```
    private byte[] GetMergedPdfBytes(byte[] fdfBytes, out string err)

err = string.Empty;
FdfReader fdfReader = null;
MemoryStream memStream = null;
PdfReader pdfReader = null;
PdfStamper stamper = null;
byte[] pdfBytes = null;
try
{
   fdfReader = new FdfReader(fdfBytes);
   Trace.Write("Read Fdf bytes into the reader");
   string writeFdf = ConfigurationManager.AppSettings["WriteFdf"];
   if (writeFdf == "true")
   {
      Trace.Write("Writing Fdf bytes to a file");
      string fdfPath = AppDomain.CurrentDomain.BaseDirectory +
                  "\\App_Files\\MyFDF.fdf";
      FileStream fs = File.Create(fdfPath);
      BinaryWriter bw = new BinaryWriter(fs);
      bw.Write(fdfBytes);
      bw.Close();
      fs.Close();
      Trace.Write("Fdf bytes written to: " + fdfPath);
   }
   Trace.Write("Merging Pdf data..");
   byte[] pdfData = null;
   string pdfHost = fdfReader.FileSpec;
   Trace.Write("Pdf location is: " + pdfHost);
   System.Net.WebClient client = new System.Net.WebClient();
   Trace.Write("Downloading Pdf bytes..");
   Uri addr = new Uri(pdfHost, UriKind.RelativeOrAbsolute);
   pdfData = client.DownloadData(addr);
   if (pdfData == null)
   {
      throw new Exception("Could not get Pdf data.");
   }
   Trace.Write("Downloading complete");
   // check if the PDF is password protected
   MiscUtility miscUtils = new MiscUtility();
   string fileName = HttpUtility.UrlDecode(Path.GetFileName(pdfHost));
   Trace.Write("Checking if Pdf is password protected");

   byte[] password = miscUtils.GetDocumentPassword(pdfData, fileName);
   if (password == null)
   {
      Trace.Write("Pdf not password protected.");
      pdfReader = new PdfReader(pdfData);
   }
```


Copyright © 2016 eSign Emcee

```
else
{
   Trace.Write("Pdf password protected. Opening using the
             password.");
   pdfReader = new PdfReader(pdfData, password);
}

pdfReader.ConsolidateNamedDestinations();
pdfReader.RemoveUnusedObjects();
Trace.Write("Merging the FDF fields into the PDF");
memStream = new MemoryStream();

stamper = new PdfStamper(pdfReader, memStream);
stamper.FormFlattening = true;
//initialize script to make fields transparent
//string js = "";
//js += " var i = 0;" + "\r";
//js += " var fldName = '';" + "\r";
//js += " var oFld = null;" + "\r";
//js += " for(i=0;i<this.numFields;i++)" + "\r";
//js += " {" + "\r";
//js += "   fldName = this.getNthFieldName(i);" + "\r";
//js += "    oFld = this.getField(fldName);" + "\r";
//js += "    if (oFld != null) {" + "\r";
//js += "     if ((oFld.type == 'checkbox') || (oFld.type ==
                'radiobutton')) {" + "\r";
//js += "       oFld.style = style.cr;" + "\r";
//js += "       oFld.textColor = color.blue;" + "\r";
//js += "       oFld.fillColor = color.transparent;" + "\r";
//js += "       oFld.readonly = true;" + "\r";
//js += "     }" + "\r";
//js += "    }" + "\r";
//js += " }" + "\r\r";
//js += " this.dirty = false;";
//stamper.JavaScript = js;

AcroFields fields = stamper.AcroFields;
Hashtable map = fields.Fields;
string[] keys = new string[map.Count];
map.Keys.CopyTo(keys, 0);
//iterate all the controls and set it printable
Trace.Write("Setting field properties to visible/hidden");
foreach (string key in keys)
{
   //assign blank value for all text fields
   //if no value assigned, iText does not make the text area transparent
   //bug in iText?
   if (fields.GetFieldType(key) == AcroFields.FIELD_TYPE_TEXT)
      fields.SetField(key, "");
```


Copyright © 2016 eSign Emcee

```
    //we don't flatten signature, radio button, check box for transparency
    if (!((fields.GetFieldType(key) ==
           AcroFields.FIELD_TYPE_SIGNATURE) ||
          (fields.GetFieldType(key) ==
           AcroFields.FIELD_TYPE_RADIOBUTTON) ||
          (fields.GetFieldType(key) ==
           AcroFields.FIELD_TYPE_CHECKBOX) ||
          (key == "pdf_passthrough") || (key == "pdf_majver") ||
          (key == "pdf_minver")))
           stamper.PartialFormFlattening(key);

    if ((fields.GetFieldType(key) !=
         AcroFields.FIELD_TYPE_PUSHBUTTON) &&
        (key != "pdf_passthrough") &&
        (key != "pdf_majver") &&
        (key != "pdf_minver"))
         fields.SetFieldProperty(key, "flags",
         PdfAnnotation.FLAGS_PRINT, null);
}
fields.SetFields(fdfReader);
Trace.Write("Merging the FDF fields into the PDF complete");
// Show or hide the CHECK/RADIO boxes based on the
   selected/unselected state.
//show if the field is selected and vice versa.
// For Radio boxes show or hide is at the group level, i.e. all of
   the fields are showed or hided.
foreach (string key in keys)
{
    if ((fields.GetFieldType(key) ==
         AcroFields.FIELD_TYPE_CHECKBOX) ||
        (fields.GetFieldType(key) ==
         AcroFields.FIELD_TYPE_RADIOBUTTON))
    {
        Trace.Write("The field '" + key + "' value is " +
        fields.GetField(key));
        if (fields.GetField(key).ToLowerInvariant() == "off")
        {
            fields.SetFieldProperty(key, "flags",
            PdfAnnotation.FLAGS_HIDDEN, null);
        }
    }
}
// Close the Pdf Reader
pdfReader.Close();
pdfReader = null;
// Close the Pdf Stamper
stamper.Close();
stamper = null;
// Close the Fdf Reader
fdfReader.Close();
fdfReader = null;
```


Copyright © 2016 eSign Emcee

```
    // Read the Pdf bytes from the memory stream
    pdfBytes = memStream.ToArray();
    // Close the memory stream
    memStream.Close();
    memStream = null;
    Trace.Write("Merging Pdf data complete. Returning Pdf bytes");
    return pdfBytes;
}
catch (Exception ex)
{
    err = ex.Message;
    Trace.Warn(err);
    return null;
}
finally
{
    // Close the Pdf Reader
    if (null != pdfReader)
    {
        pdfReader.Close();
        pdfReader = null;
    }
    // Close the Pdf Stamper
    if (null != stamper)
    {
        stamper.Close();
        stamper = null;
    }
    // Close the Fdf Reader
    if (null != fdfReader)
    {
        fdfReader.Close();
        fdfReader = null;
    }
    // Close the memory stream
    if (null != memStream)
    {
        memStream.Close();
        memStream = null;
    }
}
```

4.2 Create XML formatted additional data

To create a signing ceremony eSign Emcee requires some additional data apart from the PDF/FDF document. As specified earlier in the document the source of data can be anything. It can be a database, an XML, the PDF itself or a combination of one or all of the given sources. What the eSign Emcee Web Service expects is that this data be given in a specific format. This format is XML based and the structure has been explained below.

In the current implementation some of these values are read from PDF itself in a hidden field named “pdf_passthrough” and rest from the XML file named “defaultvalues.xml”.

Copyright © 2016 eSign Emcee

The following code snippet illustrates how to read the ceremony meta data from the PDF [contained in the "pdf_passthrough"] and modify the ceremony XML. The code needs to be added to the "Page_Load" event in the page

```
 null;
equires additional data (apart from the Pdf/Fdf data)
 a signing ceremony
e for this additional data can be anything (a database, XML etc.)
ional data is stored (for this example) in a default XML file
ult XML file this create the XML document to be sent to the WS
eation of signing ceremony
ultValuesXMLPath =
urationManager.AppSettings["DefaultValuesXMLPath"];
ilePath = Server.MapPath(defaultValuesXMLPath);
e("Default values XML path is: " + xmlFilePath);
sts(xmlFilePath))

w Exception("Could not locate the default values
    file.");


e("Reading Passthrough Data");
itUtils = new iTextUtility();
nPdf(pdfData);
assFieldValue =
GetFieldValue("pdf_passthrough");
ePdf();
FieldValue == null || pdfPassFieldValue == "" ||
FieldValue == string.Empty)

w Exception("Could not read pass through values.");

e("Passthrough Data is: " + pdfPassFieldValue);
e("Parsing Passthrough Data");

ht = ParsePassThroughData(pdfPassFieldValue, out err);
ll)
w Exception(err);

e("Recreating XML data.");

ata = RecreateXMLData(ht, xmlFilePath, out err);
 == null)
w Exception(err);
e Pdf/Fdf data to base64 formatted string
 the WS expects the Pdf/Fdf data in string format
e("Reading Pdf data");
lfData = Convert.ToBase64String(pdfData);
ata == null)
w Exception("Could not convert PDF data to string.");
```

The following code illustrates parsing the ceremony meta data defined in ""pdf_passthrough"" field of the PDF along with recreation of ceremony XML.

 Copyright © 2016 eSign Emcee

```
ivate Hashtable ParsePassThroughData(string dataToParse, out string err)

 err = null;
 try
 {
     Hashtable ht = new Hashtable();
     string[] nameValuePairs = dataToParse.Split(';');
     if (nameValuePairs.Length != 0)
     {
         Trace.Write("Passthrough name value pairs: ");
         foreach (string nvp in nameValuePairs)
         {
             string[] individualNVP = nvp.Split(':');
             if (individualNVP.Length == 2)
             {
                 ht.Add(individualNVP[0], individualNVP[1]);
                 Trace.Write(individualNVP[0] + " = " + individualNVP[1]);
             }
         }
     }
     // add the Pdf file name to the list by reading the name from the URL
     string docHost = Request.UrlReferrer.AbsoluteUri;
     Trace.Write("URI: " + docHost);
     string docName = docHost.Substring(docHost.LastIndexOf("/") + 1);
     Trace.Write("Document Name: " + docName);
     if (docName.EndsWith(".pdf") == true || docName.EndsWith(".fdf") == true)
     {
         ht.Add("PdfFileName", HttpUtility.UrlDecode(docName));
         Trace.Write("PdfFileName = " + HttpUtility.UrlDecode(docName));
     }
     return ht;
 }
 catch (Exception ex)
 {
     err = "Could not parse ceremony data. Malformed data was provided.";
     Trace.Write("Exception in ParsePassThroughData() method: " + ex.Message);
     return null;
 }
```


Copyright © 2016 eSign Emcee

```
ivate string RecreateXMLData(Hashtable passthroughData, string xmlFilePath,
                     out string err)

const string agentIDXPath = "SigningCeremony/AgentID";
const string clientIDXPath = "SigningCeremony/ClientID";
const string ceremonyTypeXPath = "SigningCeremony/CeremonyType";
const string ceremonyNameXPath = "SigningCeremony/CeremonyName";
const string ceremonyDescXPath = "SigningCeremony/CeremonyDescription";
const string documentNameXPath = "SigningCeremony/DocumentName";
const string documentDescXPath = "SigningCeremony/DocumentDescription";
const string pdfFileNameXPath = "SigningCeremony/PdfFileName";
const string genricErrorMessage = "Could not parse ceremony data. Malformed
                         data was provided.";
err = null;
string passthroughValue = string.Empty;
StringWriter writer = null;
try
{
   XmlDocument xmlDoc = new XmlDocument();
   xmlDoc.Load(xmlFilePath);
   passthroughValue = passthroughData["AgentID"].ToString();
   if (passthroughValue == string.Empty || passthroughValue == null)
   {
      Trace.Write("AgentID was not provided.");
      throw new Exception(genricErrorMessage);
   }
   XmlNode node1 = xmlDoc.SelectSingleNode(agentIDXPath);
   node1.InnerText = passthroughData["AgentID"].ToString();
   Trace.Write("Changed AgentID to " + passthroughData["AgentID"].ToString());

   passthroughValue = passthroughData["ClientID"].ToString();
   if (passthroughValue == string.Empty || passthroughValue == null)
   {
      Trace.Write("ClientID was not provided.");
      throw new Exception(genricErrorMessage);
   }
   XmlNode node2 = xmlDoc.SelectSingleNode(clientIDXPath);
   node2.InnerText = passthroughData["ClientID"].ToString();
   Trace.Write("Changed ClientID to " +
            passthroughData["ClientID"].ToString());

   passthroughValue = passthroughData["CeremonyType"].ToString();
   if (passthroughValue == string.Empty || passthroughValue == null)
   {
      Trace.Write("CeremonyType was not provided.");
      throw new Exception(genricErrorMessage);
   }
   XmlNode node3 = xmlDoc.SelectSingleNode(ceremonyTypeXPath);
   node3.InnerText = passthroughData["CeremonyType"].ToString();
   Trace.Write("Changed CeremonyType to " +
            passthroughData["CeremonyType"].ToString());
```


Copyright © 2016 eSign Emcee

```
passthroughValue = passthroughData["CeremonyName"].ToString();
if (passthroughValue == string.Empty || passthroughValue == null)
{
   Trace.Write("CeremonyName was not provided.");
   throw new Exception(genricErrorMessage);
}
XmlNode node4 = xmlDoc.SelectSingleNode(ceremonyNameXPath);
node4.InnerText = passthroughData["CeremonyName"].ToString();
Trace.Write("Changed CeremonyName to " +
          passthroughData["CeremonyName"].ToString());

string ceremonyDesc = null;
if (passthroughData.ContainsKey("CeremonyDescription"))
{
   if (passthroughData["CeremonyDescription"].ToString().Trim() ==
      string.Empty ||
      passthroughData["CeremonyDescription"].ToString() == null)
      ceremonyDesc = passthroughData["CeremonyName"].ToString();
   else
      ceremonyDesc = passthroughData["CeremonyDescription"].ToString();
}
else
   ceremonyDesc = passthroughData["CeremonyName"].ToString();

XmlNode node5 = xmlDoc.SelectSingleNode(ceremonyDescXPath);
node5.InnerText = ceremonyDesc;
Trace.Write("Changed CeremonyDescription to " + ceremonyDesc);

passthroughValue = passthroughData["DocumentName"].ToString();
if (passthroughValue == string.Empty || passthroughValue == null)
{
   Trace.Write("DocumentName was not provided.");
   throw new Exception(genricErrorMessage);
}
XmlNode node6 = xmlDoc.SelectSingleNode(documentNameXPath);
node6.InnerText = passthroughData["DocumentName"].ToString();
Trace.Write("Changed DocumentName to " +
          passthroughData["DocumentName"].ToString());

string documentDesc = null;
if (passthroughData.ContainsKey("DocumentDescription"))
{
   if (passthroughData["DocumentDescription"].ToString().Trim() ==
      string.Empty ||
      passthroughData["DocumentDescription"].ToString() == null)
      documentDesc = passthroughData["DocumentName"].ToString();
   else
      documentDesc = passthroughData["DocumentDescription"].ToString();
}
else
   documentDesc = passthroughData["DocumentName"].ToString();

XmlNode node7 = xmlDoc.SelectSingleNode(documentDescXPath);
node7.InnerText = documentDesc;
Trace.Write("Changed DocumentDescription to " + documentDesc);
```


Copyright © 2016 eSign Emcee

```
    passthroughValue = passthroughData["PdfFileName"].ToString();
    if (passthroughValue == string.Empty || passthroughValue == null)
    {
        Trace.Write("PdfFileName was not provided.");
        throw new Exception(genricErrorMessage);
    }
    XmlNode node8 = xmlDoc.SelectSingleNode(pdfFileNameXPath);
    node8.InnerText = passthroughData["PdfFileName"].ToString();
    Trace.Write("Changed PdfFileName to " +
    passthroughData["PdfFileName"].ToString());

    Trace.Write("Regeneration complete. Saving file...");
    // write the modified XML in a string
    writer = new StringWriter();
    xmlDoc.Save(writer);
    string xmlData = writer.ToString();
    Trace.Write("XML data is :" + xmlData);
    return xmlData;
}
catch (Exception ex)
{
    err = genricErrorMessage;
    return null;
}
finally
{
    if (null != writer)
        writer.Close();

}
```

4.2.1 The XML structure

4.2.1.1 Signing Ceremony <SigningCeremony />

<SchemaVersion />	Schema version
<AgentID />	The eSign Emcee User to whom the ceremony will be assigned
<ClientID />	A unique Client Id. issued by eSign Emcee to authenticate clients
<CreatorUsername />	eSign Emcee designer user name
<CreatorPassword />	eSign Emcee designer password
<StartPageEnabled />	Flag indicating if the start page should be displayed
<EndPageEnabled />	Flag indicating if the end page should be displayed
<CeremonyType>	Client defined 'Type' to which the template belongs. E.g. Credit Cards, Consumer Deposits, Loans etc.
<CeremonyName />	The name of the ceremony.
<CeremonyDescription />	A brief description of the ceremony
<SigningOrder />	The signing order to enforce (By Page = 0, By Signer = 1 or Any Order = 2).
<DocumentName />	The name of the Document.
<DocumentDescription />	A brief description of the document.
<PdfFileName />	The name of the input PDF file name (with extension).
<TimeSpan />	The document time-span. Should be an integer value between

Copyright © 2016 eSign Emcee

	0 and 99 (0-99).
<Units />	Integer value indicating the time-span units (Hours = 0, Days = 1, Weeks = 2, Months = 3 or Years = 4).
<HandBackType />	The type of handback (PUSH = 0 or PULL = 1, OFF = 2).
<HandBackMode />	Integer value indicating the mode to use in transfer (FTP = 0, e-mail = 1, File Copy = 2).
<HandBackLocation />	The location where the documents are to be handed back. e.g FTP URL for FTP, Directory path for File Copy
<HandBackUserName />	FTP user name to use in the handback
<HandBackPassword />	FTP password to use in the handback
<HandBackEmails />	One or more email addresses where the handback notifications are to be sent.
<ReturnURL />	The URL to display once the handback is triggered
<IndexMapper />	Handback Index file creation information.
<SignaturePrefixes />	Signature field prefixes to identify the method of signature.

Copyright © 2016 eSign Emcee

4.2.1.2 Index File Creation <IndexMapper />

<RootElement />	The root element to be displayed in the index XML file.
<Form name="" />	The PDF from which the values are to be extracted. The name of the PDF should be specified in the 'name' attribute.
<TagName />	The XML tag to be displayed. The form field name from which the value for the tag is to be extracted must be specified within these tags.

Note: These are generic index file specifications. If custom index file specifications are used then the above elements can be omitted. A blank <IndexMapper /> can given.

4.2.1.3 Signature Prefixes <SignaturePrefixes />

<DigitalSignature />	The signature field name prefix to match to make the signature of the method 'Digital'
<EpadSignature />	The signature field name prefix to match to make the signature of the method 'Epad'
<OptionalSignature />	The signature field name prefix to match to make the signature of the method 'Epad Optional'.
<Default />	The default signature method if none of the above given prefixes match.

Copyright © 2016 eSign Emcee

4.2.2. XML Sample

```
<?xml version="1.0" encoding="utf-8"?>
<SigningCeremony>
      <!--The schema version.-->
      <SchemaVersion>1</SchemaVersion>
      <!--The Agent (user) to whom the ceremony will be assigned -->
      <AgentID>888801</AgentID>
      <!--A unique Client Id. issued by Emcee to authenticate clients -->
      <ClientID>299D1C37-E6A8-4e1e-AD90-4B84BBF5C7E3</ClientID>
      <!--The username of the template creator.-->
      <CreatorUsername />
      <!--The password of the template creator.-->
      <CreatorPassword />
      <!--Landing Pages (Start Page, End Page, Both, None).-->
      <StartPageEnabled>false</StartPageEnabled>
      <EndPageEnabled>false</EndPageEnabled>
      <!--Type to which the template belongs.-->
      <!--e.g. Credit Cards, Consumer Deposits, Loans etc..-->
      <CeremonyType>Loans</CeremonyType>
      <!--The name of the template.-->
      <CeremonyName>Demo Session</CeremonyName>
      <!--A brief description of the template.-->
      <CeremonyDescription>
      Demo Session with Handback Implementation
      </CeremonyDescription>
      <!--The signing order to enforce-->
      <!--(By Page = 0, By Signer = 1 or Any Order = 2).-->
      <SigningOrder>1</SigningOrder>
      <!--The name of the Document.-->
      <DocumentName>Interlink_Electronics Demo Form</DocumentName>
      <!--A brief description of the document.-->
      <DocumentDescription>
      Document with 7 pages and 4 signatures (ePad, Optional and Digital signatures)
      </DocumentDescription>
      <!--The name of the pdf file(with extension) which is uploaded.-->
      <PdfFileName>Demo_Blank_Dig.pdf</PdfFileName>
      <!--The timespan by which the document shoud be completed.-->
      <!--The timespan  (0-99).-->
      <TimeSpan>3</TimeSpan>
      <!--The timespan units -->
      <!--(Hours = 0, Days = 1, Weeks = 2, Months = 3 or Years = 4).-->
      <Units>3</Units>
      <!--The hand back process - typically involves handing back the documents-->
      <!--after the signing ceremony is completed-->
      <!--Type can be set to PUSH = 0 or PULL = 1, Default is PULL -->
      <HandBackType>0</HandBackType>
      <!--The mode used in hand back. e.g FTP = 0, e-mail = 1, File Copy = 2,
           Default is e-mail.-->
      <HandBackMode>2</HandBackMode>
```


Copyright © 2016 eSign Emcee

```
        <!--The location where the documents are to be handed back. -->
        <!--e.g FTP URL for FTP, Directory path for File Copy.-->
        <HandBackLocation>C:\HandbackFiles</HandBackLocation>
        <!--The Username if FTP is used for the hand back process.-->
        <HandBackUserName></HandBackUserName>
        <!--The Password if FTP is used for the hand back process.-->
        <HandBackPassword></HandBackPassword>
        <!--The e-mail addresses to notify the handback status. -->
        <!--or to handback files as attachment in case mode is e-mail-->
        <!--Multiple entries in case multiple notifications are to be made. -->
        <!--At least one e-mail is to be provided for notifications. -->
        <HandBackEmails />
              <Email>test1@goodlife.com</Email>
              <Email>test2@goodlife.com</Email>
        </HandBackEmails>
        <!--The URL of the page to be displayed once handback is fired. -->
        <ReturnURL>http://yoursite.com/Return/Default.htm</ReturnURL>
        <!--Information to create the index.xml file. -->
        <!--'name' attribute will contain documents (PDF forms) from which the values
        are to extracted. -->
        <!--The field names are expected to be unique. -->
        <!--The elements between the 'IndexMapper' will be copied as is. -->
        <!--Only the 'FieldName*' will be replaced by their corresponding values. -->
        <IndexMapper />
        <!--The signature field name prefixes to determine the method of signature.-->
        <!--The method is determined by the starting characters of the signature
        field.-->
        <SignaturePrefixes>
              <DigitalSignature>DIG</DigitalSignature>
              <EpadSignature>REQ</EpadSignature>
              <OptionalSignature>OPT</OptionalSignature>
              <!--If the application is not able to match the above prefixed.-->
              <!--The default method is applied to all the fields.-->
              <!--The default methods can be Digital Signature = 0, Epad Signature = 1,
              Optional Signature = 4-->
              <Default>0</Default>
        </SignaturePrefixes>
</SigningCeremony>
```


Copyright © 2016 eSign Emcee

4.3 Configure the Web.config

Next step is to call to the web service to create a signing ceremony. The method returns base64 encoded Ceremony Id. The Ceremony Id is usefully to make other eSign Emcee Web Services calls such as to generate audit reports, checking ceremony status etc. The method also returns an authentication ticket which enables you to login to eSign Emcee Presenter directly without having to do a formal login for ceremony execution.

The eSign Emcee Web Service requires an authentication key to create a session. This authentication key can be stored in the Web.config file. The path (URL) to the “LaunchPresenter.aspx” should also be stored in the Web.config file. Launch Presenter helps to authenticate and “Launches” the “Presenter” to enable you to complete the signing ceremony. The following snippet demonstrates the keys added to the Web.config.

The domain name key is used to authenticate if the request to the proxy is coming from a known/trusted domain. This is to avoid proxy requests from unknown applications.

```
.
.
<appSettings>
      <add key="AuthKey" value="4EA58C9F-AEC2-4878-90D3-0AE6A96B7607"/>
      <add key="PresenterPath"
      value="http://host/IntegriSignEmceePresenter/LaunchPresenter.aspx
      "/>
      <add key="DomainName" value="yourdomain" />
</appSettings>
```

Note: There may be other keys not listed here. These keys could be used for other application configurations.

 Copyright © 2016 eSign Emcee

4.4 Call the eSign Emcee Web Service

Once the keys have been added to the Web.config file we make a call to the eSign Emcee Web Service to create a signing ceremony.

```
try
{
    // covert the Pdf/Fdf data to base64 formatted string
    // because the WS expects the Pdf/Fdf data in string format
    Trace.Write("Reading Pdf data");
    string strPdfData = Convert.ToBase64String(pdfData);
    if (strPdfData == null)
        throw new Exception("Could not convert PDF data to string.");

    Trace.Write("Calling web service");
    string wsErr = string.Empty;
    string sessionKey = string.Empty;
    string authKey = string.Empty;
    // create an instance of the WS
    EmceeWebService.EmceeServices _es = new EmceeWebService.EmceeServices();
    Trace.Write("Created WS instance");
    sessionKey = ConfigurationManager.AppSettings["AuthKey"];
    Trace.Write("Key: " + sessionKey);
    if (sessionKey == string.Empty || sessionKey == "")
        throw new Exception(wsErr);
    // create the signing ceremony by passing the Pdf/Fdf data and the XML
       containing the
    // additional data required for Emcee to create a signing ceremony
    // the method returns an authentication ticket that can be used to open a
       signing ceremony
    string cerId = string.Empty;
    Trace.Write("Creating ceremony..");
    cerId = _es.CreateSigningCeremonyFromDocument(sessionKey, strPdfData,
                                      xmlData, out authTicket, out wsErr);
    Trace.Write("Ceremony created. Ceremony Id is:" + cerId);
    if (authTicket == null)
    {
        throw new Exception(wsErr);
    }
    Trace.Write("Web serice call finished");
}
catch (...)
{
    // handle exceptions here
}
```


Copyright © 2016 eSign Emcee

4.5 Launch the eSign Emcee "Presenter"

Once the signing ceremony is created we use the authentication key returned by the eSign Emcee Web Service to "Launch" a signing session. To do this we need to call the "LaunchPresenter.aspx" in eSign Emcee Presenter and pass the authentication key as a query string parameter. The following code illustrates this.

```
.
Trace.Write("Redirecting to presenter");
string presenterPath = ConfigurationManager.AppSettings["PresenterPath"];
Response.Redirect(presenterPath + "?AuthTicket="
                                + HttpUtility.UrlEncode(authTicket));
.
.
```

 Copyright © 2016 eSign Emcee