

PDF Viewer Component V5

Edraw PDF Viewer Component is a light weight ActiveX Control which enables your application to display and interact with PDF files. It is identical to Adobe Reader program and adds high speed viewing of PDF documents to your applications easily. Read-Only Support for PDF Viewer! It is a great solution for companies wishing to display read only PDF document to their employees while restricting save or save to the underlying data. The control is lightweight and flexible, and gives developers new possibilities for using PDF Reader in a custom solution.

Home Page: <http://www.edrawsoft.com/pdfviewer.php>

Online Demo: <http://www.ocxt.com/pdfdemo.php>

Support Mail: support@edrawsoft.com

Sales online: <http://www.edrawsoft.com/purchase-pdf-component.php>

It can be easily integrated into applications written in languages that support ActiveX control such as Visual C++, Visual Basic, Delphi, C++ Builder and .Net languages. You can find the sample projects in the install folder.

Name: PDF Viewer Component

CLSID: 44A8091F-8F01-43B7-8CF7-4BBA71E61E04

Version: 5,0,0,412

Release Date: 2015-01-28

How to add PDF Viewer Component to your Visual Basic 6.0 project

1. From the Project Menu select **Components...**
 2. Select control "**PDF Viewer Component**"
 3. Click the OK Button
 4. The control will now appear in your toolbox
 5. Drag and drop the control on your form
- There are some methods and properties to operator the component.

How to add PDF Viewer Component to your ASP.NET project

1. Open Visual Studio.
2. Create an ASP.NET project.
3. Don't attempt to add the PDF Viewer Component to the Toolbox. It's a client component. You can add it as the HTML Object.
4. Copy Default.aspx and Default.aspx.vb to your project.
5. Add exist items...
6. Review the sample project in the install folder.

How to add PDF Viewer Component to your .NET project

1. Open Visual Studio .NET
2. Right-click on the toolbox and select "Add/Remove Items..."
3. Select the COM Components Tab
4. Check PDF Viewer Component and click OK Button
5. The control should appear in the toolbox as "PDF Viewer Component"
6. Double-click on the control to add to your form
7. Resize and move the control on the form as desired
8. Add a button to the form
9. Double click on the button to access code editor and enter the following code within the Click event: `PDFViewer1.LoadFile("http://www.ocxt.com/demo/samples/sample.pdf")`
10. Run the application and click on the button. The PDF should appear!

Code a solution using the control

The control is very customizable. You can change the color scheme of any of the control elements, as well as determine the border type and a custom caption. These can be set at run time or design time as needed.

Methods:

1. Load PDF File

```
Function void LoadFile (BSTR strPath);
```

Description: Opens a PDF file located on a drive or remote web server.

```
PDFViewer1.LoadFile "http://www.ocxt.com/demo/samples/pdfviewerref.pdf"  
'Or PDViewer1.LoadFile "c:\test.pdf"
```

2. HTTP Open File From Stream

```
Function: HRESULT HttpOpenFileFromStream([in] BSTR WebUrl, [in, optional] VARIANT  
WebUsername, [in, optional] VARIANT WebPassword);
```

Description: Opens a file from a stream with the HTTP/HTTPS.

Parameters:

WebUrl: A string containing the web url from which downloads data via HTTP.

WebUsername: A string containing the user name.

WebPassword: A string containing the access password.

Return Value:

Nonzero if successful; otherwise 0.

Either you want to open an appointed file or open a file from database, for client side, all what you need do is the same, like following:

```
<script language="vbscript">
Sub DownloadFile()
    PDFViewer1.HttpInit();
    PDFViewer1.HttpAddpostString(L"DocumentID", L"Test.pdf");
    PDFViewer1.HttpOpenFileFromStream(strDownloadPath, varOpt, varOpt);
End Sub
</script>
```

Before you call function `HttpOpenFileFromStream`, you should do two things, one is to initialize http for clearing all parameters and cookies in http, another thing is to appoint the file or database record. And then use `HttpOpenFileFromStream` to send the request to the destined webpage. Before `HttpOpenFileFromStream` send request, it will add a couple of parameters automatically.

`m_OAHttp.AddPostArgument(L"EDA_GETSTREAMDATA", L"EDA_YES");` This couple of parameters tell the destined webpage PDFViewer will received file as stream.

At the web side, webpage will decide to read which file or database record according to the post parameters. And you should add boundary flag 'EDA_STREAMBOUNDARY' to file data, following is the asp.net demo.

```
if (Request.Params["EDA_GETSTREAMDATA"] == "EDA_YES")
{
    String fullFileName = Server.MapPath(Request.Params["DocumentID"]);
    Byte[] fs = File.ReadAllBytes(fullFileName);
    Response.Write("Get Stream Successfully!");
    Response.Write("EDA_STREAMBOUNDARY");
    Response.BinaryWrite(fs);
    Response.Write("EDA_STREAMBOUNDARY");
}
```

asp:

```
If I_bWrStreamBoundary
Then Response.Write "EDA_STREAMBOUNDARY"
While Not I_stream.EOS And Response.IsClientConnected
Response.BinaryWrite(I_stream.Read(I_nChunkSize))
Wend I_stream.Close
If I_bWrStreamBoundary
Then Response.Write EDA_STREAMBOUNDARY"
```

3. Print PDF File

Function: boolean PrintOut ();

Description: Prints the PDF file.

4. Save Documents

Function: `boolean SaveCopyAs ()`;

Description: Saves the opened file to specify a save location.

5. Close Document

Function: `boolean Clear()`;

Description: Close the currently open document.

6. Is Adobe Reader Installed

Function: `boolean AdobeReaderIsInstalled ()`;

Description: Returns whether the client installs the PDF Reader.

7. Get Adobe Reader Version

Function: `BSTR GetVersions ()`;

Description: Returns the version number of the Adobe PDF Reader.

8. Go to the First Page

Function: `boolean GotoFirstPage ()`;

Description: Goes to first page of the PDF document.

9. Go to the Previous Page

Function: `boolean GotoPreviousPage ()`;

Description: Goes to previous page of the PDF document.

10. Go to the Next Page

Function: `boolean GotoNextPage()`;

Description: Goes to next page of the PDF document.

11. Go to the last Page

Function: `boolean GotoLastPage()`;

Description: Goes to last page of the PDF document.

12. Go to Page

Function: `boolean GotoPage(long n)`;

Description: Goes to the page n of the PDF document.

13. GoForwardStack

Function: **boolean** GoForwardStack ();
Description: Forwards to the next viewing step.

14. GoBackwardStack

Function: **boolean** GoBackwardStack();
Description: Retraces to previous viewing step.

15. ApplyPageMode

Function: **boolean** ApplyPageMode (BSTR pageMode);
Description: If the computer installed the Adobe Reader, the method will return true. Otherwise return false.
Params:
Possible values: "none": displays the document, but does not display bookmarks or thumbnails (default) "bookmarks": displays the document and bookmarks "thumbs": displays the document and thumbnails.

16. ApplyLayoutMode

Function: **boolean** ApplyLayoutMode (BSTR layoutMode);
Description: Sets the layout mode for a page view according to the specified string.
Params:
Possible values: "DontCare": use the current user preference "SinglePage": use single page mode (as it would have appeared in pre-Acrobat 3.0 viewers) "OneColumn": use one-column continuous mode "TwoColumnLeft": use two-column continuous mode with the first page on the left "TwoColumnRight": use two-column continuous mode with the first page on the right.

17. ApplyNamedDest

Function: **boolean** ApplyNamedDest (BSTR nameDest);
Description: Changes the page view to the named destination in the specified string.
Params: The named destination to which the viewer will go.

18. PrintWithDialog

Function: **boolean** PrintWithDialog ();
Description: Prints the document according to the options selected in a user dialog box. The options include embedded printing (printing within a bounding rectangle on a given page), as well as interactive printing to a specified printer. This method returns immediately, even if the printing has not completed. NOTE: If security settings do not allow printing, this method will be ignored.

19. ApplyZoom

Function: **boolean** ApplyZoom (float percent);
Description: Set the zoom factor for document page view.
Params: Specify the zoom factor in %.

20. ApplyZoomScroll

Function: **boolean** ApplyZoomScroll (float percent, float left, float top);
Description: Sets the zoom and scroll factors, using float or integer values. For example, a scale value of 100 indicates a zoom value of 100%.
Scroll values left and top are in a coordinate system where 0,0 represents the top left corner of the visible page, regardless of document rotation.

21. ApplyView

Function: **boolean** ApplyView (BSTR viewMode);
Description: Sets the view of a page according to the specified string.
Params: Possible values: "Fit": fits the entire page within the window both vertically and horizontally. "FitH": fits the entire width of the page within the window. "FitV": fits the entire height of the page within the window. "FitB": fits the bounding box within the window both vertically and horizontally. "FitBH": fits the entire width of the bounding box within the window. "FitBV": fits the entire height of the bounding box within the window.

22. ApplyViewRect

Function: **boolean** ApplyViewRect(float left, float top, float width, float height)
Description: Sets the view rectangle according to the specified coordinates.
Param:
left - The upper left horizontal coordinate.
top - The vertical coordinate in the upper left corner.
width - The horizontal width of the rectangle.
height - The vertical height of the rectangle.

23. PrintPages

Function: **boolean** PrintPages(long from, long to);
Description: Prints the specified pages without displaying a user dialog box. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed. NOTE: If security settings do not allow printing, this method will be ignored.
Param:
from - The page number of the first page to be printed. The first page in a document is page 0.
to - The page number of the last page to be printed.

24. PrintPagesFit

Function: **boolean** PrintPagesFit(long from, long to, boolean shrinkToFit);

Description: Prints the specified pages without displaying a user dialog box. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed. NOTE: If security settings do not allow printing, this method will be ignored.

Param:

from - The page number of the first page to be printed. The first page in a document is page 0.

to - The page number of the last page to be printed.

shrinkToFit - Specifies whether the pages will be shrunk, if necessary, to fit into the imageable area of a page in the printer.

25. PrintAll

Function: **boolean** PrintAll ();

Description: Prints the entire document without displaying a user dialog box. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed. NOTE: If security settings do not allow printing, this method will be ignored.

26. PrintAllFit

Function: **boolean** PrintAllFit(boolean shrinkToFit);

Description: Prints the entire document without displaying a user dialog box, and the pages are shrunk, if necessary, to fit into the imageable area of a page in the printer. The current printer, page settings, and job settings are used. This method returns immediately, even if the printing has not completed. NOTE: If security settings do not allow printing, this method will be ignored.

Param: Specifies whether the pages will be shrunk, if necessary, to fit into the imageable area of a page in the printer.

27. ApplyCurrentHighlight

Function: **boolean** ApplyCurrentHighlight(long a, long b, long c, long d);

Description: Highlights a specified rectangle on the displayed page. Use the page command before this command.

The rectangle values are integers in a coordinate system where 0,0 represents the top left corner of the visible page, regardless of document rotation.

28. HTTP Upload File

Function: **HRESULT** HttpInit();

Description: Initializes the HTTP connection.

Function: HRESULT HttpAddpostString([in] BSTR Name, [in] BSTR Value);
Description: Adds the post parameter.
Parameters:
Name: The parameter name.
Value: The parameter value.

Function: HRESULT HttpAddPostFile([in] BSTR LocalFilePath, [in] BSTR NewFileName);
Description: Adds the post file.
Parameters:
LocalFilePath: The full file path needs to upload to the server. If the parameter is NULL, the function will add the file which is opening in the component.
NewFileName: The new file name for the upload file.

Function: HRESULT HttpPost([in] BSTR WebUrl,[in, optional] VARIANT WebUsername, [in, optional] VARIANT WebPassword);
Description: Sends the specified request to the HTTP server.
Parameters:
WebUrl: A string containing the web url from which uploads data via HTTP.
WebUsername: A string containing the user name.
WebPassword: A string containing the access password.

Return Value:
Nonzero if successful; otherwise 0.

The follow code is demo how to upload the opened file to server with the HTTP mode. It can also post multiple files in a post request.

```
<script language="vbscript">  
Sub UploadFile()  
    PDFViewer1.HttpInit  
    PDFViewer1.HttpAddpostString "author", "anyname"  
    PDFViewer1.HttpAddpostString "Data", "2008-10-6"  
    PDFViewer1.HttpAddPostFile "c:\\test.pdf"  
    PDFViewer1.HttpPost "http://localhost:1320/Samples/UploadAction.aspx"  
End Sub  
</script>
```

Note: If you try to save the opened file to remote server with the "HTTP" methods, you need write a receipt page in your web server. Because the component uploads the file by HTTP mode. The follow is some sample code.

ASP.NET:

```
//Default.aspx  
<script language="vbscript">  
Sub SavetoServer()  
PDFViewer1.HttpInit
```



```
PDFViewer1.HttpAddpostString "author", "anyname"
PDFViewer1.HttpAddpostString "Data", "2007-5-15"
PDFViewer1.HttpAddPostOpenedFile
PDFViewer1.HttpPost "http://localhost:1320/Samples/UploadAction.aspx"
'PDFViewer1.Save "http://localhost:1320/UploadAction.aspx?author=name&Data=2"
End Sub
</script>

//UploadAction.aspx.cs file
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Xml;
using System.Drawing.Imaging;
using System.Text.RegularExpressions;

public partial class UploadAction : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (Request.QueryString["author"] == "name" && Request.QueryString["Data"] ==
"2")
        {
            Response.Write("0\n");
            Response.Write("We have receipted the right param from PDF ActiveX Control.");
        }
        if (Request.Files.Count == 0)
        {
            Response.Write("0\n");
            Response.Write("There isn't file to upload.");
            Response.End();
        }
        if (Request.Files[0].ContentLength == 0)
        {
            Response.Write("0\n");
            Response.Write("Failed to receipt the data.\n\n");
            Response.End();
        }
    }
}
```

```
    }
    string fullFileName = Server.MapPath(Request.Files[0].FileName);
    Request.Files[0].SaveAs(fullFileName);
    Response.Write("Upload Successfully.");
    Response.End();
}
}
```

PHP:

```
<?php
header("http/1.1 200 OK");
$user = iconv("UTF-8", "UNICODE", $_POST['user']);
$password = iconv("UTF-8", "UNICODE", $_POST['passwd']);
$sql = sprintf("username=%s passwd=%s", $user,$password);
echo $sql;
$sql = sprintf("file=%s size=%s error=%s tmp=%s",
$_FILES['trackdata']['name'],$_FILES['trackdata']['size'],$_FILES['trackdata']['error'],$_FILES[
'trackdata']['tmp_name']);
echo $sql;
$handle = fopen($_FILES['trackdata']['name'],'w+');
if($handle == FALSE)
{
    exit("Create file error!");
}
$handle2 = fopen($_FILES['trackdata']['tmp_name'],'r');
$data = fread($handle2,$_FILES['trackdata']['size']);
echo $data;
fwrite($handle,$data);
fclose($handle2);
fclose($handle);
exit(0);
?>
```

ASP: (review the full code in the install folder\samples\asp\)

```
<%@Language=VBScript %>
<!-- #include file="./include/upload.inc" -->
<!--#include file="./include/conn.asp"-->
<%
Set Uploader = New UpFile_Class
Uploader.NoAllowExt="cs;vb;js;exe"
Uploader.GetData (Request.TotalBytes)
Request.TotalBytes
if Uploader.isErr then
    select case Uploader.isErr
        case 1
```

```
Response.Write "Fail to receipt the data."
case 2
Response.Write "The file is too big to upload"
End select
'Response.End
End if
Dim id
If "" <> Request.QueryString("id") then
    id = Request.QueryString("id")
End if
if id<>0 then
    Sql="SELECT * from doc where doc.id = "&id
else
    Sql="SELECT * from doc"
End if
rs.Open Sql,conn,1, 3
for each formName in Uploader.file
    set file=Uploader.file(formName)
    If id<>0 then
        If("" = Request.QueryString("isAip")) Then
            rs("DocContent") = Uploader.FileData(formname)
            rs("DocID") = Uploader.Form("DocID")
            rs("DocTitle") = Uploader.Form("DocTitle")
            rs("DocType") = Uploader.Form("DocType")
        Else rs("AipContent") = Uploader.FileData(formname)
            rs("state") = 2
        End if
    Else
        rs.AddNew
        rs("DocID") = Uploader.Form("DocID")
        rs("DocTitle") = Uploader.Form("DocTitle")
        rs("DocContent") = Uploader.FileData(formname)
        rs("Docdate") = Now()
        rs("DocType") = Uploader.Form("DocType")
        rs("state") = 1
    End If
set file=nothing
rs.Update
exit for
next
rs.Close
Set rs=Nothing
conn.close
set conn = Nothing
```

```
Set Uploader = Nothing
%>
```

JSP:

```
jsp:
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page language="java" import="com.jspsmart.upload.File">
<jsp:useBean id="mySmartUpload" scope="page" class="com.jspsmart.upload.SmartUpload"
/>
<%
String sPath="C:\\\"
mySmartUpload.initialize(pageContext);
mySmartUpload.upload();
String TempName=mySmartUpload.getRequest().getParameter("TempName");
mySmartUpload.save(sPath);
File myFile =mySmartUpload.GetFiles().getFile(0);
%>
```

29. HTTP Download File

Function: **HRESULT** HttpDownloadFileToTempDir([in] BSTR WebUrl, [in, optional] VARIANT WebUsername, [in, optional] VARIANT WebPassword, [out,retval] VARIANT*vFilePath);

Description: Downloads a file from a remote server then save it to OA temporary directory with HTTP.

Parameters:

WebUrl: A string containing the web url from which downloads data via HTTP.

WebUsername: A string containing the user name.

WebPassword: A string containing the access password.

Return Value:

Local temporary file path to save the download file.

The component provides the method to download file from a server then save to a local disk file. Support Http and HTTPS.

```
<script language="vbscript">
Sub DownloadFile()
    Dim sPath;
    sPath = PDFViewer1.HttpDownloadFileToTempDir "http://www.ocxt.com/demo/1.pdf"
    MsgBox sPath
End Sub
</script>
```

Note: You should make sure you the file exists in the web site firstly. A simple method is to enter the **WebUrl** in the Internet Explorer. If IE can download the file, the component can do it too.

FAQ: If you are using Windows 2003 as your server, you maybe need to add the mime types to

Internet Information Server.

30. HTTP Open File From Server Directory

Function: HRESULT HttpOpenFileFromServerDir([in] BSTR WebUrl, [in, optional] VARIANT WebUsername, [in, optional] VARIANT WebPassword);

Description: Opens a file from a remote server directory with HTTP/HTTPS.

Parameters:

WebUrl: A string containing the web url from which downloads data via HTTP.

WebUsername: A string containing the user name.

WebPassword: A string containing the access password.

Return Value:

Nonzero if successful; otherwise 0.

The component provides the method to download file from a server then save to a local disk file. Support Http and HTTPS.

```
<script language="vbscript">
Sub DownloadFile()
    PDFViewer1.HttpOpenFileFromServerDir "http://www.ocxt.com/demo/sample.pdf"
End Sub
</script>
```

31. FTP Upload/Download File

Function: HRESULT FtpConnect([in] BSTR WebUrl, [in, optional] VARIANT WebUsername, [in, optional] VARIANT WebPassword);

Description: Creates a FTP connect.

Function: HRESULT FtpDownloadFile([in] BSTR RemoteFile, [in] BSTR LocalFile);

Description: Download a file from remote server then save it to local file with FTP.

Parameters:

RemoteFile: The remote file path which saves to FTP site.

LocalFile: The full file path which downloads to the local disk.

Function: HRESULT FtpUploadFile([in] BSTR LocalFile, [in] BSTR RemoteFile, [in, optional] VARIANT OverWrite);

Description: Uploads a local disk file to remote server with FTP.

Parameters:

LocalFilePath: The full file path needs to upload to the server. If the parameter is NULL, the function will add the file which is opening in the component.

RemoteFile: The remote file path which saves to FTP site.

OverWrite: Overwrites the existed file if the same file exists at the FTP server.

Function: HRESULT FtpDisconnect();

Description: Closes the FTP Connect.

Return Value:

Nonzero if successful; otherwise 0.

The follow code is demo how to download a file to local disk with the FTP mode.

```
<script language="vbscript">
Sub UploadFileviaFTP()
    PDFViewer1.FtpConnect "ftp.ocxt.com", "username", "password"
    PDFViewer1.FtpDownloadFile "ftp.ocx.com/archieve/001.pdf", "c:\temp.pdf"
    PDFViewer1.FtpDisconnect
End Sub
</script>
```

32. Set Read Only

Function: void SetReadOnly();

Description: Protects the current open PDF document.

1. Hide the File toolbar
2. Disable the Copy button
3. Hide the Right click menu
4. Disable the Save, Print, Copy, Show/Hide Toolbar hot key.

Tips:

If you want to set read only property when you load a pdf file, you need put the method at OnDocumentOpened event.

```
<SCRIPT ID=clientEventHandlersJS LANGUAGE=javascript>function
PDFViewer1_NotifyCtrlReady()
{//document.PDFViewer1.LicenseName = "yourlicensename";
//document.PDFViewer1.LicenseKey = "yourlicensename";
document.PDFViewer1.LoadFile ("e:\\OA_Reference2.pdf");
}
function PDFViewer1_OnDocumentOpened(FileName)
{
document.PDFViewer1.SetReadOnly();
}
</SCRIPT>
<SCRIPT LANGUAGE=javascript FOR=PDFViewer1 EVENT=NotifyCtrlReady>
<!--
```

```
PDFViewer1_NotifyCtrlReady();  
//-->  
</SCRIPT>  
<SCRIPT LANGUAGE=javascript FOR=PDFViewer1 EVENT=OnDocumentOpened(FileName)>  
<!--  
PDFViewer1_OnDocumentOpened(FileName);  
//-->  
</SCRIPT>
```

33. Disable Toolbars

Function: VARIANT_BOOL DisableToolbar([in] VARIANT_BOOL bDisable);

Description: Disables all the toolbars in the Adobe Reader. The method must be call after the PDF file opened.

34. Disable Toolbars Right Click Menu

Function: VARIANT_BOOL DisableToolbarRightClickMenu([in] VARIANT_BOOL bDisable);

Description: Disables the right click menu at the toolbars. The method must be call after the PDF file opened.

35. Disable Toolbars Right Click Menu

Function: VARIANT_BOOL DisableViewRightClickMenu([in] VARIANT_BOOL bDisable);

Description: Disables the right click menu at the view. The method must be call after the PDF file opened.

36. Disable Toolbars Right Click Menu

Function: VARIANT_BOOL DisableViewRightClickMenu([in] VARIANT_BOOL bDisable);

Description: Disables the right click menu at the view. The method must be call after the PDF file opened.

37. Disable Hotkeys

Function: VARIANT_BOOL DisableViewRightClickMenu([in] VARIANT_BOOL bDisable);

void DisableHotKeyPrint();

void DisableHotKeySave();

```
void DisableHotKeyCopy();
void DisableHotKeyShowBookMarks();
void DisableHotKeyShowThumbnails();
void DisableHotKeyShowToolbars();
void DisableHotKeySearch();
```

38. Get a Temporary Path

```
VARIANT_BOOL GetTempFilePath();
```

Description: Gets a temporary file path.

39. Clear Temporary Path

```
VARIANT_BOOL ClearTempFiles();
```

Description: Clears these temporary files created by the component. The function won't delete any other files out of OA temporary directory.

Properties:

1. File Path

Property: **BSTR** Path

Description: The property will save the open file path.

You can use the property to decide whether the component opened a PDF file.

```
If PDFViewer1.Path != "" Then
PDFViewer1.ApplyZoom 0.75
End If
```

2. Show or hide the toolbars

Property: **boolean** Toolbars

Description: Show/Hide whether toolbars should be displayed.

You can open a PDF document without toolbars as follow:

```
If OA1.GetToolbars = True Then
OA1.oobars False
Else
OA1.Toolbars True
End If
```

3. Show or hide the navigation panes

Property: **boolean** NavPanes

Description: Show/Hide whether navigation panels should be displayed.

4. Show or hide the messages

Property: **boolean** Messages

Description: Show/Hide whether messages should be displayed.

5. Show or hide the scrollbar

Property: **boolean** Scrollbar

Description: Show/Hide whether scrollbar should be displayed.

6. Show or hide the statusbar

Property: **boolean** Statusbar

Description: Show/Hide whether statusbar should be displayed.

7. Set the Parameters for the Open Document.

Property: **BSTR** NamedDest;

BSTR Page;

BSTR Comment;

BSTR Zoom;

BSTR View;

BSTR ViewRect;

BSTR PageMode;

BSTR Search;

BSTR Highlight;

Description: These properties have the same function with the above methods. If you special the Zoom as 75, the component will open any PDF document with the default 75% view zoom. If you special the Search as "Flowchart", the component will highlight the Flowchart word when it opens a PDF document.

8. Modify the Border Style

Property: **short** BorderStyle;

Description: Set the caption of the titlebar.

The property allows the developer to modify the border style. It can be the follow value:

```
typedef enum BorderStyle
```

```
{
    BorderNone = 0,
    BorderFlat,
    Border3D,
    Border3DThin
```

```
} BorderStyle;
```

```
<object classid="clsid:053AFEB4-D968-435F-B557-19FF76372B1B" id="EDrawPDFViewer1"
width="657" height="452">
    <param name="BorderStyle" value="2">
</object>
```

9. Set the Color Scheme

Properties:

OLE_COLOR BorderColor;

OLE_COLOR BackColor;

OLE_COLOR ForeColor;

OLE_COLOR TitlebarColor;

OLE_COLOR TitlebarTextColor;

Description: Set the color scheme for the component.

```
<object classid="clsid:053AFEB4-D968-435F-B557-19FF76372B1B" id="EDrawPDFViewer1"
width="657" height="452">
    <param name="BorderColor" value="-2147483632">
    <param name="BackColor" value="-2147483643">
    <param name="ForeColor" value="-2147483640">
    <param name="TitlebarColor" value="-2147483635">
    <param name="TitlebarTextColor" value="-2147483634">
</object>
```

10. Modify the TitlebarText

Property: **BSTR** TitlebarText;

Description: Set the caption of the titlebar.

The property allows the developer to modify the caption in the titlebar.

```
<object classid="clsid: 44A8091F-8F01-43B7-8CF7-4BBA71E61E04" id="PDFViewer1"
width="657" height="452">
    <param name=" TitlebarText " value="PDF Viewer Component">
</object>
```

11. Set the Color Scheme

Property: **BSTR** LicenseName
Description: Gets/Sets the license name.

12. License Key

Property: **BSTR** LicenseKey
Description: Gets/Sets the license key.

Events:

1. Ready to Open Document Event

Event: HRESULT NotifyCtrlReady();
Description: Ready to open document.

When the component finished the initialization the NotifyCtrlReady event is raised. The event allows you to open a new document after the component has released all the resources are ready to open a new document.

HTML + JScript

```
<html>
<script language=javascript id=clientEventHandlersJS>
function NotifyCtrlReadyEvent ()
{
    PDFViewer1.Open "http://www.ocxt.com/demo/sample.pdf"
}
</script>

<body>
<object classid="clsid: 44A8091F-8F01-43B7-8CF7-4BBA71E61E04 " id="PDFViewer1"
width="657" height="452">
    <param name="Toolbars" value="0">
</object>
</body>

<script language="JScript" for=PDFViewer1 event=" NotifyCtrlReady ()">
    NotifyCtrlReadyEvent ();
</script>
</html>
```

2. Document Opened Event

Event: void OnDocumentOpened([in] BSTR FileName);
Description: Called when document is opened or new document added.

Every time a user opened a file successfully, the OnDocumentOpened event is raised. The event allows you to override the default behavior for the control and supply your own custom actions and dialog boxes to do normal file operations.

HTML + JScript

```
<script language="JScript" for=PDFViewer1 event=" OnDocumentOpened(FileName)">
OnDocumentOpenedEvent(FileName);
</script>
<script language=javascript>
function OnDocumentOpenedEvent(FileName)
{
    Alert("Respond the Document Opened Event.");
}
</script>
```

3. Document Close Event

Event: `void OnDocumentClosed();`

Description: Called when document is closed.

Every time a user closed a file successfully, the OnDocumentClosed event is raised. The event allows you to override the default behavior for the control and supply your own custom actions and dialog boxes to do normal file operations.

HTML + JScript

```
<script language="JScript" for= PDFViewer1 event=" OnDocumentClosed()">
OnDocumentClosedEvent();
</script>
<script language=javascript>
function OnDocumentClosedEvent()
{
    Alert("Respond the Document Close Event.");
}
</script>
```

4. Before Document Closed Event

Event: `void BeforeDocumentClosed([in,out] VARIANT* Cancel);`

Description: Called before document is closed (may be canceled).

Every time before a user closed a document the OnActivationChange event is raised. The event allows you to override the default behavior for the control and supply your own custom actions and dialog boxes to do normal file operations.

HTML + JScript

```
<script language="JScript" for= PDFViewer1 event=" BeforeDocumentClosed (Cancel)">
OnBeforeDocumentClosedEvent(GoingActive);
</script>
```

```
<script language=javascript>
function OnBeforeDocumentClosedEvent (GoingActive)
{
    Alert("Respond the Before Document Closed Event.");
}
</script>
```

5. Before Document Saved Event

Event: **void** BeforeDocumentSaved([in,out] VARIANT* Cancel);

Description: Called before document is saved (may be canceled).

Every time a user opens a new document or closes a document the OnActivationChange event is raised. The event allows you to override the default behavior for the control and supply your own custom actions and dialog boxes to do normal file operations.

HTML + JScript

```
<script language="JScript" for= PDFViewer1 event=" BeforeDocumentSaved( Cancel)">
OnBeforeDocumentSavedEvent(GoingActive);
</script>

<script language=javascript>
function OnBeforeDocumentSavedEvent ( Cancel)
{
    Alert("Respond the Before Document Saved Event.");
}
</script>
```

6. Before Right Click the Window

Event: **HRESULT** OnWindowBeforeRightClick();

Description: Called before right click the component. The event needs the DisableViewRightClickMenu method or DisableToolbarRightClickMenu method is called.

7. Before Double Click the Window

Event: **HRESULT** OnWindowBeforeDoubleClick ();

Description: Called before double click the component.

8. Before Download File

Event: **HRESULT** BeforeDownloadFile ();

Description: Called before downloading the file.

9. Download File Completed

Event: **HRESULT** DownloadFileComplete ();

Description: Called when the file was downloaded completely.

System Requirements

Hardware :

Minimum:

Pentium Based MMX processor 500 MHZ

256 MB of ram

SVGA Graphics card

64 MB HardDisk

Recommended:

Intel Celeron or AMD Duron 1 GHz and above

1 GB ram

SVGA Graphics card with some acceleration (1024x768x16bitsColor)

Software :

Minimum:

Windows 2000 with IE6 and above.

Recommended:

IE 6/7/8

Windows 2000/XP/2003/Vista/Windows 7/8/8.1

Adobe PDF Reader 6/7/8/9/X/XI

For web application: The component need IE Protection Mode was Turn Off.

You can added your site at the IE trusted site list to turn off the IE protection mode automatically when the users visit your site with the component.

Please consider also your application's requirements.

Redistribute Files

Here are the files needed for installing PDF Viewer Component.

DLL and OCX

PDFViewer.ocx

PDFViewerX.dll

Or you can use the cab file in the install folder for web application.

PDFViewer.cab

Visual Basic Issue

How to add PDF Viewer Component to your Visual Basic 6.0 project

1. From the Project Menu select Components...
2. Select control "PDF Viewer Component Module" in the controls table.
3. Click the OK Button.
4. The control will now appear in your toolbox.
5. Drag and drop the control on your form.
6. Right click the control then choose the View Code... item.
7. Add the Form_Load event.

```
Private Sub Form_Load()  
PDFViewer1.LicenseName = ""  
PDFViewer1.LicenseKey = ""  
End Sub
```

Add a button and fire the button click event.

```
Private Sub OpenDoc()  
PDFViewer1.Open "d:\1.pdf"  
End Sub
```

8. Run the project.

Visual C++ Issues

Adding the control to a simple dialog base Application

1. Begin a new MFC AppWizard(exe) dialog base Application.
2. Open The Resources Dialogs.
3. Select the Main Dialog and right click on it.
4. From pop-up menu select "Insert ActiveX Control...".
5. Select PDF Viewer Component from the list.
6. Add a member variable to newly created control in the dialog class.
7. This will automatically generate new cpp and .h files which including the information of PDFViewer control.
8. Call the MFC ClassWizard to add the control event message

To resize the control according the form add a new Window Message Handler WM_SIZE and add the following code in OnSize virtual function

```
if(!::IsWindow(m_PDFViewer.m_hWnd)) return;
```

```
m_PDFViewer.MoveWindow(0,0,cx,cy);
```

See pdfviewer.exe c++ example

Upgrade a control in VC++

You can easily upgrade a VC++ project witch is using a PDF Viewer control of a previous version:

1. Register the new version of control using the regsvr32 utility that is located in windows system directory.
2. From Project menu select Add to Project -> Components and Controls
3. From Components and Control Gallery dialog select the folder Registered ActiveX Controls
4. Find and select PDF Viewer Component and click OK to all next dialogs.
5. Rebuild the project.

Web Application Issue

Use the pdfviewer.cab file

The component can be used for web application. You can embed it at the html Object tag.

```
<object classid="clsid:44A8091F-8F01-43B7-8CF7-4BBA71E61E04"
id="PDFViewer" width="100%" height="100%"
codebase="http://www.yoursite.com/download/pdfviewer.cab#5,0,0,412">
</object>
```

Note: You should put the officeviewer.cab file in your own site and change the codebase url when you distribute the component.

You can view the html samples in the install folder\samples\html folder.

The PDFViewer.cab file is available in the intall folder. You can create your own cab file.

Why do I fail to download the ActiveX control on the client machine

The failure of loading ActiveX control has the following possible causes:

1. The security settings of IE on the client machine are incorrect.

Please verify the following security settings of IE to "Prompt" or "Enabled":

- a) Download signed ActiveX controls
- b) Run ActiveX Controls and plug-ins
- c) Script ActiveX controls marked safe for scripting

The dialog box of the security setting can be launched from menu Tools>Internet Options. Then select the security tab.

How to add Edraw Component to your ASP.NET project

1. Open Visual Studio.
2. Create a new ASP.NET project.

3. Don not attempt to add the PDF Viewer Component to the Toolbox. It is a client component. You can add it as the HTML Object.
4. Copy all files at the ASP_c#\ to the new project folder. (UploadAction.aspx UploadAction.aspx.cs Default.aspx.cs Default.aspx Tester.doc)
5. Then add exist items...
6. Modify the Server Port in the Default.aspx.
6. Run.

About the IE Protection Mode

The component can't run at the IE protection mode. So the client needs add the site in IE trusted site list. The component will pop up the reminder dialog if it run at the IE protection mode.

How to Delete the IE CAB Download

For IE7 or IE8

1. Open a new Internet Explore.
2. Go to Internet Explorer, and click the "Tools" button in the left of browser, and then click Manage Add-ons.
3. Click Toolbars and Extensions.
4. Double Click the add-on you want to delete. In the pop up message box, you can click the Delete button in the bottom.

For IE6

IE Toolbar > Options > General > Setting > View Object...

Then delete the PDF Viewer Component.

Firefox Support

Microsoft created Active X for the Internet Explorer browser to properly play various types of media. Although the tool was designed specifically for Internet Explorer, when using Firefox, you will sometimes come across a website instructing you to install Active X to view media properly. For that reason, Mozilla has created a plug-in that will allow Active X to be enabled in Firefox.

Instructions

1. Determine what version of Firefox you are using by clicking on "Help" from the Firefox menu bar. Click "About Firefox" to see the version number listed underneath the Firefox logo.
2. Visit the Firefox Mozilla website to locate the [Esker Active X Plug-In](#) (see Resources).

3. Read to see which version of the plug-in is compatible with the version of Firefox you are running. If there is more than one version of Active X for your Firefox version, always download the latest Active X version since it will reflect updates.
4. Ensure that the box is checked to install the "Experimental Add-On." Click "Add to Firefox."
5. Accept the license agreement by clicking on the "Accept and Install" button. Once the automatic installation completes, Active X will be activated automatically for Firefox. If for some reason it does not auto-activate, click on "Tools" and "Add-Ons" from the Firefox menu. Click the "Plug-Ins" tab and click "Enable."

Convert to Full Version

The trial version has 60 day limitation.

After you put the order, you will get the full version download link and the license key.

Step:

1. Firstly you need uninstall the trial version from your computer. Make sure the pdfviewer.ocx file has been removed from your computer completely.
2. Install the full version. In the install folder, you can find all files to redistribute.
3. For VC, VB, C# desktop application, you can set LicenseName and LicenseKey property directly in the Property panel.
4. For Web Application, you need set the two properties in the NotifyCtrlReady event.

```
<script language="javascript">
function PDFViewer_NotifyCtrlReady()
{
document.PDFViewer.LicenseName = "your license name";
document.PDFViewer.LicenseKey = "your license code";
}
</script>
<SCRIPT LANGUAGE=javascript FOR=PDFViewer EVENT=NotifyCtrlReady>
<!--
PDFViewer_NotifyCtrlReady();
//-->
</SCRIPT>
<object classid="clsid: 44A8091F-8F01-43B7-8CF7-4BBA71E61E04"
id="PDFViewer" width="100%" height="100%"
codebase="http://www.yoursite.com/download/pdfviewer.cab#3,2,0,380">
</object>
```
5. At last you can call the PDFViewer.AboutBox method to verify the license. The trial version will show "30 day trial license for evaluation". But the full license version will show your license name.

Online Store: <http://www.edrawsoft.com/pdfviewer.php>

License Agreement: <http://www.edrawsoft.com/componentagreement.php>

Support Email: support@ocxt.com

Other ActiveX Controls

[Edraw Office Viewer Component](#) – Office Viewer Component contains a standard ActiveX control that acts as an ActiveX document container for hosting Office documents (including Microsoft Word, Microsoft Excel, Microsoft PowerPoint, Microsoft Project, and Microsoft Visio documents) in a custom form or Web page. The control is lightweight and flexible, and gives developers new possibilities for using Office in a custom solution.

[Edraw Flowchart ActiveX Control](#) – Do you want to learn how to draw? Now you can online! Learn how to draw like a local application with Edraw Flowchart ActiveX Control that lets you quickly build basic flowcharts, organizational charts, business charts, hr diagram, work flow, programming flowchart and network diagrams.