

eDocPrintPro SDK Wrapper

1. Working scenario.

The **eDocPrintPro** SDK Wrapper is a .NET assembly collection:

eDocSdkWrapper.exe – the main assembly file – remoting server and **eDocPrintPro** SDK bridge;

eDocPrintProWrapperImpl.dll – server and client functionality implementation;

eDocSDKWrapperClient.dll – the remoting client assembly and COM interface; it must be regasm registered;

Interop.EDocPort.dll – the eDocPrintPro SDK interop.

The working scenario needs the following files in the server folder:

eDocSdkWrapper.exe, eDocPrintProWrapperImpl.dll, Interop.EDocPort.dll

the following files in the remoting client folder:

eDocSDKWrapperClient.dll, eDocPrintProWrapperImpl.dll

and the following files in the application folder:

application, **eDocPrintProWrapperImpl.dll**.

2. Remoting.

The remoting uses the tcp port 7800. For the remoting events, each remoting client uses an additional tcp port, the next unused port, starting with 7801. The maximum number of allowed concurrent clients is 100. You can modify these coded numbers via the **eDocSdkWrapper.exe** command-line parameters:

/tcpport=<other port number>

/maxclients=<new number>

If you changed the tcp base port number, you must make the same change in all the clients, see the **ServerPort** property.

The remoting protocol includes a hand-shake protocol, so a dead client is removed from the server clients list. When the client list is empty, the server will quit after a configured time-out number of seconds. The default time-out is 0, it means no quit. This value can be changed through the command-line parameter **/idletimeout=<new value>**, or by a client, using the **QuitServer** method.

On the client side, the **ServerName** and **ServerPort** properties can be used to overwrite the defaults: "localhost" and 7800, before the first call of the **CreateClient** method.

In a multiserver environment, each server must provide a different port numbers range.

3. Interfaces and events.

The **eDocSDKWrapperClient.dll** provides a dual interface, with .NET and COM bearing similarities. This is an advantage of the end user application, because of the target independent functionality, but the unmanaged code must use the default .NET exported functions with care. Our advice is to only use the above documented functionality. This is divided in two logical groups: **eDocPrintPro** printer properties handling classes and **eDocPrintPro** events handling interface. Two events are fired for each printer job: **StartDocument** and **EndDocument**. Both are fired after an application creates the printing context, so modifying the printer settings inside of the event handlers is not useful. However, plug-in settings can be prepared in the **StartDocument**, and after **EndDocument** the final document is complete. During both events, the application has access to some useful information: document name, job ID, printer name, user name and machine name. Additionally, the **EndDocument** event provides a semicolon-separated files list (usually there is only one output document, but some printer settings allow page-by-page document splitting).

Because the usual implementation of the client is one client - one server or one client – many servers, the events are fired synchronously. If an application needs to show user interface or other long processing actions, it is strongly recommended to do it asynchronously, outside of the event handler.

4. The **EDocClient** class.

To work with the **eDocPrintPro** SDK wrapper, the **eDocSdkWrapper.exe** must be launched first, then an application can use the classes exposed by the **eDocSDKWrapperClient.dll**. The **eDocSdkWrapper.exe** must be launched only once for a given computer.

The main class (COM creatable) is **EDocClient**. You can use only one **EDocClient** for each server in your application. This class creates, destroys and manages a bidirectional client-server connection.

Code sample:

```
[VB6]

Dim WithEvents ewc As eDocSDKWrapperClient.EDocClient
...
Set ewc = New EDocClient
...
ewc.CreateClient
...
...
...
ewc.ReleaseClient
```

[C++]

```
eDocClient.CreateDispatch("eDocSDKWrapperClient.EDocClient");  
...  
eDocClient.CreateClient();  
...  
...  
...  
eDocClient.ReleaseClient();
```

Properties.

ClientTimeout – the number of seconds to wait until a client is removed from the clients list, if the hand-shake protocol is idle. The default value is 10. You can increase this value to avoid problems in debugging sessions.

PrinterCount – the number of eDocPrintPro printers on the server computer.

ServerName – the computer name or IP address of the server, the default “localhost”.

ServerPort – the server base port number, the default is 7800.

Methods.

CreateClient – creates a tcp bidirectional connection to a running server.

GetActionPage – returns an **ActionPage** object.

GetAddonsManager – returns an **AddonsManager** object

GetBmpPage – returns a **BmpPage** object

GetChainsManager – returns a **ChainsManager** object

GetDestinationPage – returns a **DestinationPage** object

GetDocumentPage – returns a **DocumentPage** object

GetEpsPage – returns a **EpsPage** object

GetJpgPage – returns a **JpgPage** object

GetLayoutPage – returns a **LayoutPage** object

GetPclPage – returns a **PclPage** object

GetPcxPage – returns a **PcxPage** object

GetPdfPage – returns a **PdfPage** object

GetPluginsPage – returns a **PluginsPage** object

GetPngPage – returns a **PngPage** object

GetPrinterNameFromIndex returns the printer name; the index starts with 1.

GetPsPage – returns a **PsPage** object

GetTiffPage – returns a **TiffPage** object

QuitServer – changes the server idle timeout in seconds. The server will terminate if the client list is empty after the defined number of seconds. The value of 0 means infinite timeout.

ReleaseClient – will disconnect the client from the server and remove it from the server clients list.

SetNoUIParameters – changes the printer UI parameters. You can hide the UI, except for the About pane. The method parameters allow a custom configuration for the About pane too.

5. The *Page classes.

There are several classes corresponding to the eDocPrintPro UI panes, allowing an application to retrieve and update the printer settings. Usually there is a direct correspondence between the UI fields and the *Page properties, the exceptions will be documented below.

ReadSettings – loads the printer settings in the page class. This is the first thing you must do with the page object.

WriteSettings – updates the settings with the current object properties. You can change or copy from one printer to another, using this method.

The usual scenario is illustrated in the following code sample:

[VB6]

```
Dim ap As ActionPage
Set ap = ewc.GetActionPage("eDocPrintPro")
ap.ReadSettings
ap.ActionType = ActionTypes_launchViewer
ap.WriteSettings
```

[C++]

```
CActionPage ap = eDocClient.GetActionPage("eDocPrintPro");
ap.ReadSettings();
ap.put_ActionType(1);
ap.EriteSettings();
```

Other methods.

Class **DocumentPage**, method **SetGhostscriptPaths** - allows an application to change the ghostscript version.

Class **PluginsPage**, method **GetChainName** – allows an application to enumerate the existing plug-in chains.

6. Managing add-ons.

An add-on is an **eDocPrintPro** optional part, used to create additional document types. You can only retrieve information about the installed add-ons and change some public properties for add-ons using the **AddonsManager** and **AddonItem** objects.

The **AddonsManager** is used only to enumerate the available add-ons. The property **AddonCount** and the method **GetAddon** are enough to complete the task.

The **AddonItem** class encapsulates the add-on public functionality. As design requirements, an add-on is a special dll with a few fixed elements, a transformation list and a property list. A transformation is a conversion of an **eDocPrinPro** output document into a new format. Each transformation has an input extension, an output extension and a name. The properties are pairs (name, string), specific to each add-on requirements.

ExtensionCount – property – the transformation count for the add-on.
GetExtensionName – method – the display name of the transformation
GetInputExtension – method – the input document type of a transformation.
GetOutputExtension – method - the output document type of a transformation.
PropertyCount – property - the number of the string properties
GetPropertyName – method – retrieves the name of the property
GetPropertyValue – method – retrieves the string of a property
SetPropertyValue – method – changes a property string.

7. Managing plug-ins.

A plug-in is a special assembly designated for automatic postprocessing of the **eDocPrintPro** output documents. After the installation they become available to the printer. Because more than one plug-in can be used for the same document, the user must arrange them in chains, and each chain must fix the used plug-ins and their execution order. The class **ChainsManager** allows an application to access the chains functionality.

Properties.

AskForChain – instructs **eDocPrintPro** to ask for a plug-in chain before the postprocessing.
ChainCount – the number of available chains.
CurrentChain – the current selected chain name.
PluginCount – the number of available plug-ins.

Methods.

AppendPlugin – appends a plug-in to an existing chain.
CreateChain – creates a new chain.
DeleteAllChains – deletes all defined chains.
DeleteChain – deletes a chain by name.
GetChainDescription – returns the chain description string
GetChainName – returns the chain name from index.
GetChainPluginCount – returns the number of the plug-ins in a chain.
GetChainPluginName – returns the name of a plug-in, by index.
RemoveChainPlugin – removes a plug-in from a chain.
RenameChain – changes a chain name.
SetChainDescription – changes the chain description string.
SetDeleteDocumentAfterProcessing – instructs **eDocPrintPro** to delete the document after the chain processing.
SetPositionInChain – moves a plug-in at the given position in the chain, keeping the order of the other plug-ins.