Cerenade HTML Filler Control for ASP.Net
User Manual

May 2009

Contents

Introduction

Within the past decade, the electronic forms and document management industry has been witness to the large-scale adoption of What-You-See-Is-What-You-Get (WYSIWYG) electronic forms by organizations both in the government and the private sector as a means of moving away from antiquated paper systems. These electronic form systems are typically network-based, where, for example, a server computer on a network hosts one or more electronic forms. A user, using a client computer coupled to the server computer through a network such as the Internet, or World-Wide-Web (Web), can download one of these electronic forms and submit the information requested therein to the server computer. In the world of electronic forms, ability to deliver high-fidelity WYSIWYG forms to end users without any client-side requirements has been sought by Forms Management teams in the government and the private sector for quite some time. Cerenade's patent-pending HTML Forms technology is the only solution in the Electronic Forms market that has been able to finally deliver a one-of-a-kind solution capable of delivering WYSIWYG forms to end users with zero client-side requirement – all that is required is an Internet browser on any OS platform including gaming consoles capable of browsing the Internet. Just like its thick-client counterparts, Cerenade's HTML Forms technology is capable of functionalities that are usually intrinsic only to client-side form engines; functionalities such as zooming, dynamic bar-code generation and digital signature support.

Using this manual

This Manual contains procedures for installing and operating Cerenade HTML Filler Control for ASP.Net.

This preface provides an overview of the Cerenade HTML Filler Control for ASP.Net User Manual, explaining conventions used in this manual.

Conventions

As you work with Cerenade HTML Filler Control and its manual, remember that the text format indicates a specific action or meaning. The following table lists the conventions used in this manual.

Convention	Meaning
Bold	In procedures indicates text that you type or the name of screen objects such as icons or buttons
>**Bold**	Identifies a procedure
SMALL CAPS	Refers to keys, such as SHIFT, CONTROL, or TAB
"Quotation marks"	Web links and folder names

Technical Support

You may obtain technical support for Visual eForms family of products via our Website, Phone, e-mail and fax.

Before contacting Cerenade for technical support, be sure to check our Knowledgebase on our Website for some common problems and their solutions.

Our Website also hosts our Online Download Center which allows you to obtain updates to the latest versions of Visual eForms components.

Internet:	support@cerenade.com	
Web:	www.cerenade.com	
Tech Support:	310-645-0598	8am to 5pm PST
Fax:	310-645-0599	

Installation

Installation refers to **HTML Filler Control for ASP.Net** and all the components associated with the Component.

System Requirements

The following table provides the requirements for HTML Filler Control for ASP.Net:

- Microsoft Visual Studio 2005 with AJAX Extensions

  or
- Microsoft Visual Studio 2008

Installation Procedure

The installation program is fairly self-explanatory. Simply run the setup program and follow the instructions on the screen. For redistribution of your application containing Cerenade HTML Filler Control, please refer to **Redistribution** section of this manual.

HTML Filler Control Overview

The first thing you need to know about Cerenade HTML Filler Control is that it is NOT really a standard .Net control which would find its way into Visual Studio's Toolbox after installation. Instead, it is an assembly (i.e. Cerenade.HtmlFiller.dll) along with a set of other supporting components which together as a whole provide the functionalities available in the HTML Filler Control. Without getting into too much detail, the reason behind this implementation is that for proper operation of the control, it needs full access to the `System.Web.UI.Page` object and as a matter of fact, the main constituent of the Filler Control is `Cerenade.Controls.HtmlFillerPage` which is internally derived from `System.Web.UI.Page`. That being the case, the process to integrate the Filler Control is a little more elaborate than simply dragging a component from the Visual Studio's Toolbox onto your .ASPX page, but it is certainly not that complicated of a process. The following section provides full detail on the necessary steps to use the Filler Control in an ASP.Net Application.

How to Use the Filler Control in an ASP.Net Application

In order to use the HTML Filler control in an ASP.Net application, the following steps should be taken:

1. Create a new ASP.NET Web application (if you are use Visual Studio 2005, make sure that the new application is AJAX-Enabled)

2. Add reference to **Cerenade.HtmlFiller.dll**. Note that by default, the Filler Control installation program installs this assembly in C:\Program Files\Cerenade\HtmlFiller\Components.

3. In the .ASPX page that will be responsible for displaying .FAR forms (i.e. form templates created using Cerenade Visual Designer), change it so the page is inherited from Cerenade.Controls.HtmlFillerPage instead of the default System.Web.UI.Page. Note that you can add a **Using** directive for **Cerenade.Controls** so you wouldn't have to add the prefix to **HtmlFillerPage** and its methods and properties used throughout your code.

4. From the Toolbox, add an instance of **ScriptManager** to the .ASPX page

5. For the newly added **ScriptManager**, you need to enable **PageMethods** (i.e. set **EnablePageMethods** to **true**)

6. Remove the line '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"....' from the .ASPX page

7. Create two folders which will be used for opening and displaying forms: **uploads** and **images**. Note that the names for these folders are configurable so if you wish to use other names, you may do so.

8. In **Page_Load()** event handler of the .ASPX page, add the following piece of code to tell the filler component where to place temporary background images for forms (note that these temporary images are automatically and periodically deleted):

```
if (!this.IsPostBack)
{
    this.TargetFolder = "images";
}
```

9. If you are going to use the **OpenFormDialog()** method, you need to provide the full path of the **upload** folder created in step 7. Here is an example:

```
this.OpenFormDialog(Path.Combine(Page.Request.Phys
icalApplicationPath, "uploads"));
```

Filler Control Properties

AutoCleanUp

Description: This property dictates whether temporary files and images which are created by the Filler Component are automatically and periodically deleted.

Syntax: HtmlFillerPage.AutoCleanUp

Remarks: The default value for this property is **true**.

Data Type: Boolean

ClientWidth

Description: This property is used to communicate the width of the HTML page on the Client side to the Filler Control on the server side.

Syntax: HtmlFillerPage.ClientWidth

Remarks: Note that this value should only be set the very first time a page is rendered by the Filler Control. Thereafter, the filler will automatically communicate this value from the client side to the Filler Control on the server side. If this value is not initially set, the Filler Control will assume a default value of **1000** for the very first rendition of the form page.

Data Type: String

CurrField

Description: Signifies the current field in focus.

Syntax: HtmlFillerPage.CurrField

Remarks: Use this property to Set or Get the current field in focus. Note that when assigning a value to this property, you can either use the Name or TabOrder of a Field on the form. For TabOrder, prefix the number representing the TabOrder with "@" (for example, to address a field at TabOrder 12, assign "@12" to **CurrField** property of the Filler Control.

Data Type: String

CurrPage

Description: Signifies the current page in focus.

Syntax: HtmlFillerPage.CurrPage

Remarks: Use this property to Set or Get the current page in focus.

Data Type: Int16

FieldData

Description: This indexer property provides read/write access to field values on the form.

Syntax: HtmlFillerPage.FieldData[FieldName]
where
'FieldName' can either be a literal string or a String variable containing the name of the field of interest.

Examples:

HtmlFillerPage.FieldData["Field1"] = "new value";
String FieldValue = HtmlFillerPage.FieldData["Field2"];

Remarks: Use this property to Set or Get the value of a field of interest on the currently loaded form.

Data Type: String

FieldProperty

Description: This indexer property provides read/write access to the properties of fields on the form.

Syntax: HtmlFillerPage.FieldProperty[FieldName, PropertyId]
where
'FieldName' can either be a literal string or a String variable containing the name of the field of interest and 'PropetyId' is an integer representing the field property of interest. Note that for ease of programming, the Filler Component provides Enum list Cerenade.Controls.FieldPropId containing all of the available field properties.

Examples:

```
HtmlFillerPage.FieldProperty["Field1",
      (int) FieldPropID.BACKCOLOR] = "255,0,0";

String TextColor =  HtmlFillerPage.FieldProperty["Field1",
      (int) FieldPropID.TEXTCOLOR];
```

Remarks: Use this property to Set or Get the property value of a field of interest on the currently loaded form. Note that all the property values are accessed as strings. It is therefore the responsibility of the containing application to perform the necessary conversions to other data types.

Data Type: String

FormProperty

Description: This indexer property provides read/write access to the Form properties.

Syntax: HtmlFillerPage.FormProperty[PropertyId]
where
'PropetyId' is an integer representing the form property of interest. Note that for ease of programming, the Filler Component provides Enum list Cerenade.Controls.FormPropID containing all of the available Form properties.

Examples:

HtmlFillerPage.FormProperty[(int)FieldPropID.FORMVERSION] = "v2.5";

String IndexFields = HtmlFillerPage.FieldProperty[(int)FieldPropID.INDEX_FIELDS];

Remarks: Use this property to Set or Get the Form property values. Note that all the property values are accessed as strings. It is therefore the responsibility of the containing application to perform the necessary conversions to other data types.

Data Type: String

KeepSessionAlive

Description: Dictates whether the Filler Component will keep the current Session alive which the currently loaded form is in use.

Syntax: HtmlFillerPage.KeepSessionAlive

Remarks: Setting the value of this property to **true** will automatically address the issue of losing the state of the form (e.g. the data entered on different pages on the form, etc.) in Session Timeout scenarios. The default value for this property is **true**.

Data Type: Boolean

NumPages

Description: This read-only property provides the total number of pages of the currently loaded form.

Syntax: HtmlFillerPage.NumPages

Remarks: Note that if the currently loaded form contains dynamic tables, then the total number of pages could potentially be different from the original number of pages in the form template.

Data Type: Int16

PrintPreviewMode

Description: This property provides for toggling between the Fillable state and the Print-Preview state of the form.

Syntax: HtmlFillerPage.PrintPreviewMode

Remarks: Switching to Print-Preview state allows the end-user to examine items such as non-printable objects on the form, text/fonts that have been automatically reduced in size for auto-fit purposes, etc.

Data Type: Boolean

TargetFolder

Description: This property sets or gets the folder which will contain temporary images created by the Filler Control.

Syntax: HtmlFillerPage.TargetFolder

Remarks: The value for this property has to be relative to the .NET Application Folder. Assigning an absolute path to this property will not work and it will not throw an exception either. The default value for this property is an empty String which means temporary images will be created in the main .NET Application Folder.

Data Type: String

ZoomFactor

Description: This property sets or gets the current Zoom Factor for the Filler Control.

Syntax: HtmlFillerPage.ZoomFactor

Remarks: The following settings are available for this property:

1	View the entire page in the window
2	View the entire page width in the window
3 to 200	Set the Zoom Factor to the specified number of pixels-per-inch

Data Type: String

Filler Control Methods

ClearData

Description: This method clears the data on all pages or the data on a particular page on the currently loaded form.

Syntax: void ClearData(params object[] args)

where **args** can either be empty or a page number

Remarks: None.

Examples:

```
HtmlFillerPage.ClearData();      // clear data on all pages
HtmlFillerPage.ClearData(2);     // clear data on page 2
```

CloseForm

Description: This method closes the currently loaded form.

Syntax: void CloseForm()

Remarks: Calling this method will clean-up/delete all temporary images created during the period that this form was open in the Filler Control.

Examples: HtmlFillerPage.CloseForm();

ImportDataFromFarFile

Description: This method allows for population of the currently loaded form with the data contained in another form archive (i.e. .FAR file).

Syntax: void ImportDataFromFarFile(String FormPath)

Remarks: If the form template associated with the FormPath parameter is different from the currently loaded form, then only the data from fields with common names between the two templates will be imported into the currently loaded form.

Examples: HtmlFillerPage.ImportDataFromFarFile(tempFormPath);

isFormLoaded

Description: Use this method to check if a form is currently loaded in the Filler Control.

Syntax: Boolean isFormLoaded()

Remarks: None.

Examples: Boolean b = HtmlFillerPage.isFormLoaded();

OpenForm

Description: Use this method to load a .FAR form into the Filler Control.

Syntax: void OpenForm(String Path)

Remarks: The parameter 'Path' should be an absolute path. The path can either be a URL or a local path on the hard disk. In either case, the path should be accessible to application or else, an exception will be thrown.

Examples:

```
HtmlFillerPage.OpenForm(
    Path.Combine(Page.Request.PhysicalApplicationPath,
    @"forms\SampleForm.far"));
```

OpenFormDialog

Description: Use this method to provide the end-user with a mechanism to select a .FAR form on his/her local hard disk and have it loaded into the Filler Control.

Syntax: void OpenFormDialog(String UploadFolder)

Remarks: The parameter 'UploadFolder' should be an absolute local path on the server side. Note that the call to OpenFormDialog() should generally be the last statement in the enclosing function.

Examples:

```
HtmlFillerPage.OpenFormDialog(
    Path.Combine(Page.Request.PhysicalApplicationPath,
    "uploads"));
```

PrintDialog

Description: Use this method to display a Print Dialog allowing the user to first select a Printer Destination and then print the currently loaded form based on the Print Dialog settings.

Syntax: void PrintDialog()

Remarks: Note that the call to PrintDialog() should generally be the last statement in the enclosing function.

Examples: HtmlFillerPage.PrintDialog();

PrintForm

Description: Use this method to print a form based on the desired settings provided in the method parameters.

Syntax: void PrintForm(Int16 StartPage, Int16 EndPage, String Options)

where 'Options' is a semicolon-delimited list comprised of the following Option/Value pairs:

Option	Possible Values
imageFormat	PDF | XPS | HTM
outputMode	FormAndData | FormOnly
showProgressPopUp	true | false

The default values for the above options are in order: PDF, FormAndData, false.

Remarks: Note that the call to PrintForm() should generally be the last statement in the enclosing function. Upon successful completion of the print function, the end-user will be provided with a dialog to download the printout file generated from the call to PrintForm() method.

Examples:

```
HtmlFillerPage.Print(1, 4,
        "imageFormat=PDF;outputMode=FormAndData");
```

SaveForm

Description: Use this method to save the currently load form to a .FAR file.

Syntax: void SaveForm(String FilePath, String Options)

where 'FilePath' is the local path for the .FAR file to be created and 'Options' is a semicolon-delimited list comprised of the following Option/Value pairs:

Option	Possible Values
showSaveFormPopUp	true | false

The default value for the 'showSaveFormPopUp' is false. If 'showSaveFormPopUp' is set to true, then the end-user will be provided with a dialog to download the saved form to the client computer.

Remarks: if 'showSaveFormPopUp' parameter is set to true, then the 'FilePath' parameter cannot be an absolute path (i.e. it should be a path relative to application folder

Examples:

```
HtmlFillerPage.SaveForm(this.TargetFolder + "\\dummy.far",
                        "showSaveFormPopUp=1");
```

SetOnPrintText

Description: Sets the Print-Text (i.e. Watermark) and its corresponding attributes.

Syntax:

```
void SetOnPrintText(String Text, Int32 x, Int32 y,
        String FontName, Int32 Height, Int32 Escapement,
        Boolean Bold, Boolean Italic, Boolean Underline,
        Color TextColor)
```

where method parameters are:

Parameter	Description
Text	Print-Text to be reflected on printouts
x	x Position of the text in twips (1440 twips = 1 inch)
y	y Position of the text in twips
FontName	Name of the Font to be used for Print Text
Height	Font Height in twips
Escapement	angle (in 0.1-degree units) between the escapement vector and the x-axis
Bold	Set to True for Bold Font
Italic	Set to True for Italic Font
Underline	Set to True for Underlined Font
TextColor	Color of Text as an RGB value

Remarks: Once set, the Print-Text will be reflected on all subsequent form printouts.

Examples:

```
HtmlFillerPage.SetOnPrintText("Draft", 2 * 1440, 1 * 1440,
        "Arial", 1 * 1440, 3150, true, false, false, Color.Red);
```

SignForm

Description: Signs the currently loaded form in a given Signature Field.

Syntax: void SignForm(String FieldName, String Options, Int16 Flags)

where method parameters are:

Parameter	Description
FieldName	Name or TabOrder of the Signature Field on the form. For TabOrder, prefix the number representing the Tab Order with "@"; for example, to address a field at Tab Order 12, pass "@12" as FieldName Parameter of this method.
Options	a semicolon-delimited list of Parameter/Value pairs for the following parameters: Username, Password, Domain.
Flags	Reserved for future use

Remarks: The XML-encoded data stream, if password-encrypted, is compliant with the current W3 org draft standard for XML encryption.

Examples: HtmlFillerPage.SignForm("Signature1", "Username=John;Password=mypass1;Domain=AcmeDom", 0);

XMLGetFormData

Description: Returns an XML-encoded data stream, in the form of a string, containing the Form Data.

Syntax: String XMLGetFormData(String Options)

where 'Options' is a semicolon-delimited list comprised of the following Option/Value pairs:

Option	Possible Values
EnableExtAttr	0 | 1 Setting EnableExtAttr to 1 will allow Modified properties of the fields to also be included in the returned XML string. Default value is 1.
Encrypt	0 | 1 Enables encryption of the data. Default value 0.
EncryptAlg	3DES | RC4 Encryption algorithm. Default value is 3DES.
EncryptKeyLen	[Number of Bits] Number of bits for the encryption Key. Note some algorithms such as 3DES ignore this parameter. Default value is 128.
EncryptPwd	[Password] Password used to encrypt the XML data with.
Pages	[Page Number] [Page Number] is a single page Number. If "Pages" parameter is omitted, by default all pages will be included in the returned XML string.
SkipBlanks	0 | 1 0=Don't Skip, 1=Skip If "SkipBlanks" parameter is omitted, By default only fields without empty Value will be included in the return XML string.

Remarks: The XML Schema is as follows (in DTD format):

```
<!ELEMENT FORMDATA
(VERSION,FORMNAME,FORMLOC,FORMVERSION,HEADER,ENCRYPTIO
N,FIELDDATA) >
<!ELEMENT VERSION (#PCDATA) >
<!ELEMENT FORMNAME (#PCDATA)>
<!ELEMENT FORMLOC (#PCDATA)>
<!ELEMENT FORMVERSION (#PCDATA)>
<!ELEMENT HEADER (#PCDATA)>
<!ELEMENT ENCRYPTION (#PCDATA)>
<!ELEMENT FIELDDATA (F+)>
<!ELEMENT F (#PCDATA)>
<!ATTLIST F NAME CDATA #REQUIRED>
```

Examples:

```
String XMLString = HtmlFillerPage.XMLGetFormData("");
String XMLString =
        HtmlFillerPage.XMLGetFormData("Pages=1;SkipBlanks=1");
String XMLString = HtmlFillerPage.XMLGetFormData("pages=2");
```

XMLSetFormData

Description: Populate fields on the form with data from an XML-encoded data stream.

Syntax: void XMLSetFormData(String XMLString, String Options)

where 'XMLString' is XML-encoded data stream and 'Options' is a semicolon-delimited list comprised of the following Option/Value pairs:

Option	Possible Values
EncryptPwd	[Password] Password used to decrypt the XML data with.
ClearData	0 | 1 0=Don't Clear Data,1=ClearData If "ClearData" parameter is omitted, by default all form fields are cleared prior to processing the XML String.
FormLoad	0 | 1 If "FormLoad" parameter is omitted, by default the form referenced in the XML string will be loaded prior to processing the XML string if the currently loaded form in the Filler Control is different from the one referenced by XML String.

Remarks: See "XMLGetFormData" for the XML Schema in DTD (Document Type Definition) format.

Examples:

```
HtmlFillerPage.XMLSetFormData(XMLString, "");
HtmlFillerPage.XMLSetFormData(XMLString,
                              "Formload=1;ClearData=0");
HtmlFillerPage.XMLSetFormData("XMLString, "ClearData=1");
```

Redistribution

Redistribution of your application containing Cerenade HTML Filler Control requires installation of certain components in order for your application to function properly.

What should be redistributed with your Application

HTMLFillerRedist.exe provided with this toolkit automatically installs all the necessary components required for the Cerenade HTML Filler Control to function properly on the target system where your application in installed. Note that you should run **HMTLFillerRedist.exe** prior to installing your own ASP.Net application.