IHTML

The IHTML control is used to embed server side generated HTML inside a page that is provided by an adapter property. The IHTML control is very flexible on the one hand. On the other hand, you have to take care about what is defined inside the IHTML area.

Use this control if you have, for example, a server side report generation program already producing HTML as output which you want to include into your pages, etc.

The following topics are covered below:

- Example
- Pros and Contras when Using the IHTML Control
- Scripting in Generated HTML
- Example: Building Download Links
- Properties

Example

Layout definition:

```
<rowarea name="IHTML Demo">
    <itr>
        <ihtml valueprop="generatedHTML">
        </ihtml>
    </itr>
</rowarea>
```

Program code:

```
// property >generatedHTML<
String m_generatedHTML;
public String getGeneratedHTML()
{
    return "<table border=1>"+
                "<tr>"+
                "<td width=100>1</td>"+
                "<td width=200>2</td>"+
                "<td width=50>3</td>"+
                "</tr>"+
                  "</table>";
}

public void setGeneratedHTML(String value) { m_generatedHTML = value; }
```

The above layout definition, together with the above program code, produces the following page:

Note that the IHTML control internally opens a table cell in which you place the HTML code coming from the adapter property. Your HTML must fit into this concept, i.e. it must be embeddable into an HTML arrangement like the following:

```
...
...
    <table ...>
    ...
    <tr>
        ...
        <td>
            <!-- THIS IS WHERE YOUR HTML NEEDS TO FIT IN -->
        </td>
        ...
    </tr>
    ...
    </table>
...
```

Pros and Contras when Using the IHTML Control

The IHTML control is powerful because it offers all possibilities to directly embed HTML code inside your page. The "pros" are:

- You can render complex HTML statements on the server side. The client just plugs the HTML into the page. The rendering effort on the client side is very low.
- You can directly write HTML on the server side. If you are an HTML expert, then you may like to do so in certain situations.

On the other hand, there are "cons":

- You begin to write rendering logic on your own - on the server side.
- You have to take care of being compatible through various browsers.
- You have to take care of a consistent appearance of the HTML results in order to have a consistent look and feel throughout your application.
- You may increase the traffic over the network.

Conclusion:

- HTML rendering on the server side should not be offered in general, but only under defined circumstances.
- Encapsulate your server side rendering behind a proper Java interface. Application developers work with the Java interface; the HTML rendering behind should be done by an HTML expert.

Application Designer also uses the IHTML control internally: in the area of reporting, Application Designer provides a server side Java API for simple report generation. Behind the report interface, HTML (or PDF) is generated which is passed into an internally used IHTML control.

Scripting in Generated HTML

Based on the conclusion of the previous section, you should be aware of the fact that it is also possible to add JavaScript statements into the HTML code that you define for the IHTML control. See *Customized Controls* for information on the JavaScript API of Application Designer. You can call the JavaScript library functions from the script that you pass as part of the HTML you create.

Example: you may invoke a method behind a certain link. Your program might look like this:

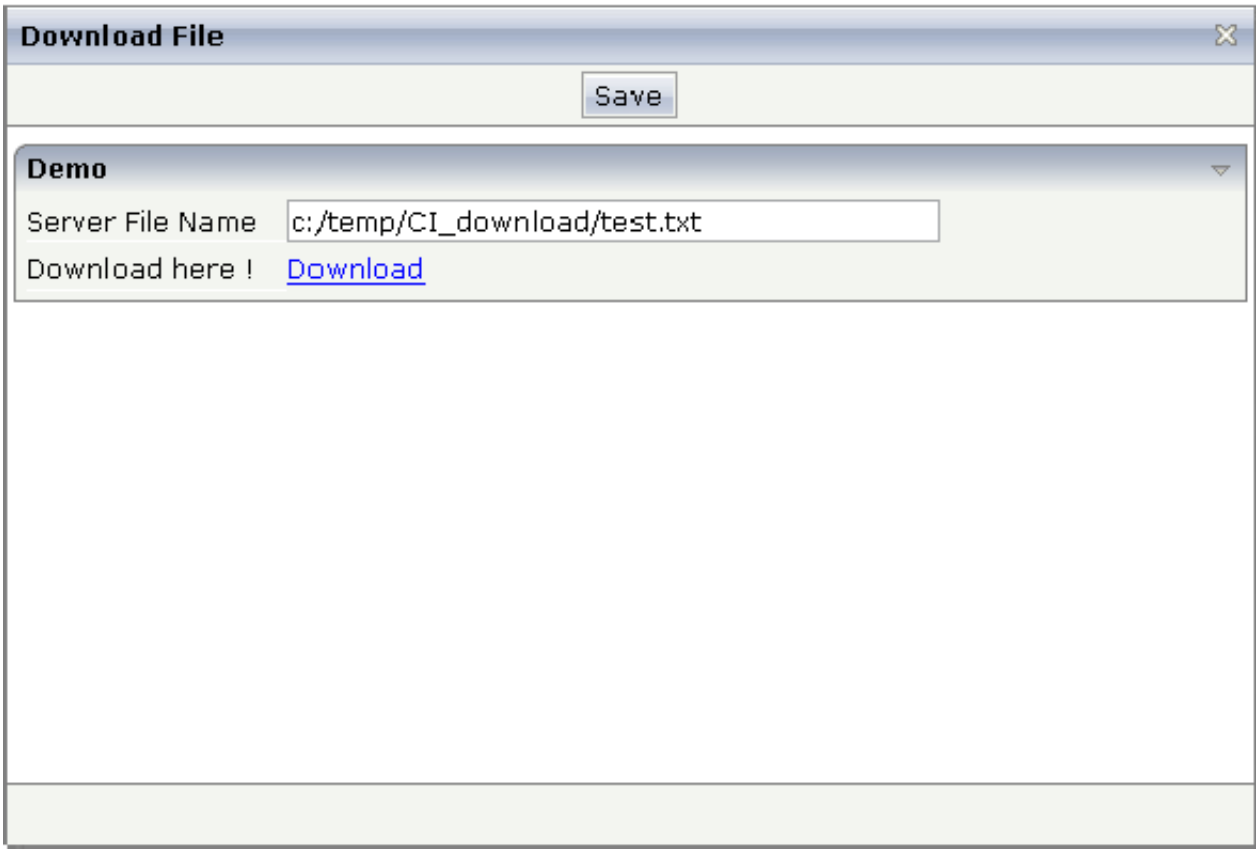
```
String m_html;

public String getHtml() { return m_html; }

public void createHTMLString()
{
    ...
    m_html += "<a href='javascript: csciframe.invokeMethodInModel(\'onClick\');'>Hello</a>";
    ...
}
```

Example: Building Download Links

While HTML provides an explicit file upload feature (which is internally used when using Application Designer's UPLOAD control), there is no explicit download control to load server data onto the client machine. However, there is the standard behavior of the browser when following a link.

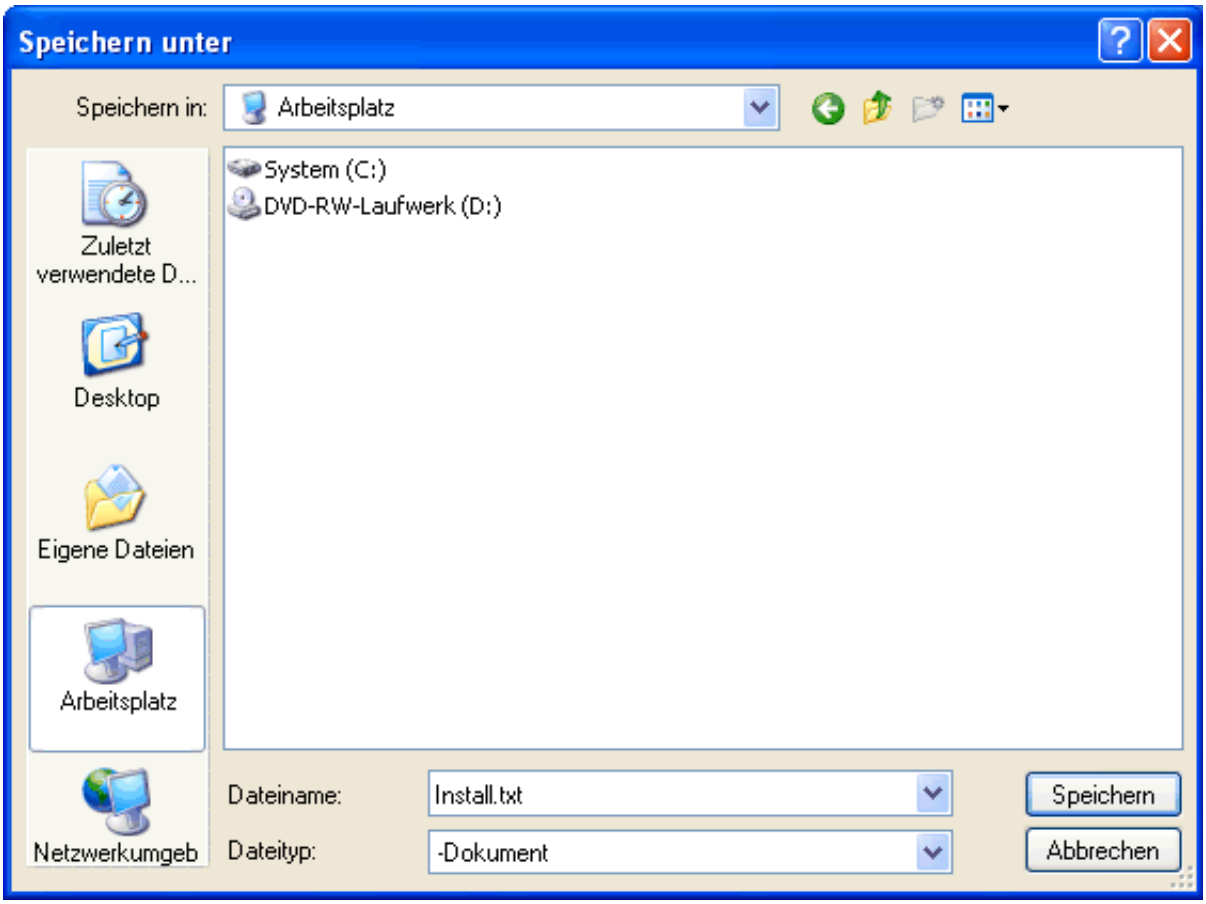
The browser knows certain document content types (MIME types). For example, the browser knows that PDF documents are displayed in an Adobe Reader plug-in. If the browser receives a URL with a content type that the browser does not understand, then it automatically opens a pop-up in which the user is asked to download the file.

Have a look at the following screen:



The user enter the name of a file that resides on the server (in this case, it is a Windows-based server). After entering the name, the user sees a **Download** link. When choosing this link, the following pop-up appears.

In the pop-up, the user is asked to save the file in the local file system of the browser. After this security message, the typical file selection dialog box appears:



The XML page layout looks as follows:

```
<page model="DownloadAdapter">
    <titlebar name="Download File">
    </titlebar>
    <header withdistance="false">
        <button name="Save">
        </button>
    </header>
    <pagebody>
        <vdist height="5">
        </vdist>
        <rowarea name="Demo">
            <itr>
                <label name="Server File Name" width="120">
                </label>
                <field valueprop="serverfilename" width="300" flush="server">
                </field>
            </itr>
            <itr>
                <label name="Download here !" width="120">
                </label>
                <ihtml valueprop="downloadURL">
                </ihtml>
            </itr>
        </rowarea>
    </pagebody>
    <statusbar withdistance="false">
    </statusbar>
</page>
```

You see that within the FIELD, the property `serverFileName` is maintained. The FIELD directly flushes value changes to the server. The HTML text that is shown inside the IHTML control, is provided by the property `downloadURL`.

The adapter code is:

```
import com.softwareag.cis.file.FileManager;
import com.softwareag.cis.server.Adapter;

// This class is a generated one.

public class DownloadAdapter
    extends Adapter
{
    // property >downloadURL<
    String m_downloadURL;
    public String getDownloadURL() { return m_downloadURL; }
    public void setDownloadURL(String value) { m_downloadURL = value; }

    // property >serverfilename<
    String m_serverfilename="c:/temp/CI_download/test.txt";
    public String getServerfilename() { return m_serverfilename; }
    public void setServerfilename(String value)
    {
        m_serverfilename = value;
    // load file
        try
        {
        // SECURITY!!! Only allow download from one certain directory
        m_serverfilename = m_serverfilename.replace('\\','/');
        if (!m_serverfilename.startsWith("c:/temp/CI_download/") &&
        !m_serverfilename.startsWith("/tmp/CI_download/"))
        {
        throw new Exception("Only can download files from dedicated directories!");
        }
        // read file content
        byte[] bytes = FileManager.readFileIntoByteArray(m_serverfilename,true);
        String contentName = m_serverfilename.replace('/','_');
        // Build logical name out of file name, this logical name is
        // the default file name that is shown in the "save as" pop-up of the browser.
        // IMPORTANT: the extension must NOT be set, otherwise the browser opens
        // the document and does not offer the download pop-up
        contentName = contentName.replace('/','_');
        contentName = contentName.replace(':','_');
        contentName = contentName.replace('.','_');
       // pass to session buffer; use content type UNDEFINED,
       // not TEXT/HTML or some existing content type!
        m_downloadURL = findCISessionContext().getSessionBuffer().
        addDocument(contentName,bytes,"UNDEFINED");
        m_downloadURL = "<a target='_blank' href='" + m_downloadURL + "'>Download</a>";
        }
        catch (Throwable t)
        {
        outputMessage(MT_ERROR,"Could not load file from server! " + t.toString());
        m_downloadURL = null;
        }
    }
}
```

When the user defines the server file name, the following things happen:

- The property `serverFileName` is set in the adapter object.
- The file name is checked so that only files of a certain server directory are accessible for download.
- The file is read on server side from the file system.
- The content of the file is transferred to a so-called session buffer. The session buffer is a small technical framework that is available for all Application Designer applications: you can store any data inside the session buffer and assign a name. The session buffer returns a URL that allows browsers to access this content. (The URL internally contains a servlet call; the servlet is the one to pick the content when being accessed by a browser.)
- The URL that is returned from the session buffer is embedded into a `<a ...>...</a>` tag, representing a link to the user.
- The property `downloadURL` that is used in the IHTML control is the one to provide the URL.

Consequence: when changing the server file name in the example, the download link will be updated and the user can always download the corresponding file. The file name that is proposed in the **Save As** dialog box is the name that you assign to the session buffer content.

For proper content type management, you see that

- a content type was chosen (`UNDEFINED`) that is not defined and is consequently treated as "unknown content",
- the name proposed for the download file does not contain an extension that is known to the browser (it does not contain an extension at all).

Properties

Basic			
valueprop	Server side property representation of the control.	Optional	

width	Width of the control. There are three possibilities to define the width: (A) You do not define a width at all. In this case the width of the control will either be a default width or - in case of container controls - it will follow the width that is occupied by its content. (B) Pixel sizing: just input a number value (e.g. "100"). (C) Percentage sizing: input a percantage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a width this control can reference. If you specify this control to have a width of 50% then the parent element (e.g. an ITR-row) may itself define a width of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 120 140 160 180 200 50% 100%
height	Height of the control. There are three possibilities to define the height: (A) You do not define a height at all. As consequence the control will be rendered with its default height. If the control is a container control (containing) other controls then the height of the control will follow the height of its content. (B) Pixel sizing: just input a number value (e.g. "20"). (C) Percentage sizing: input a percantage value (e.g. "50%"). Pay attention: percentage sizing will only bring up correct results if the parent element of the control properly defines a height this control can reference. If you specify this control to have a height of 50% then the parent element (e.g. an ITR-row) may itself define a height of "100%". If the parent element does not specify a width then the rendering result may not represent what you expect.	Optional	100 150 200 250 300 250 400 50% 100%
colspan	Column spanning of control. If you use TR table rows then you may sometimes want to control the number of columns your control occupies. By default it is "1" - but you may want to define the control to span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value

rowspan	Row spanning of control. If you use TR table rows then you may sometimes want to control the number of rows your control occupies. By default it is "1" - but you may want to define the control two span over more than one columns. The property only makes sense in table rows that are snychronized within one container (i.e. TR, STR table rows). It does not make sense in ITR rows, because these rows are explicitly not synched.	Optional	1 2 3 4 5 50 int-value
ihtmlstyle	CSS style definition that is directly passed into this control. With the style you can individually influence the rendering of the control. You can specify any style sheet expressions. Examples are: border: 1px solid #FF0000 background-color: #808080 You can combine expressions by appending and separating them with a semicolon. Sometimes it is useful to have a look into the generated HTML code in order to know where direct style definitions are applied. Press right mouse-button in your browser and select the "View source" or "View frame’s source" function.	Optional	background-color: #FF0000 color: #0000FF font-weight: bold
comment	Comment without any effect on rendering and behaviour. The comment is shown in the layout editor’s tree view.	Optional	
valign	Vertical alignment of control in its column. Each control is "packaged" into a column. The column itself is part of a row (e.g. ITR or TR). Sometimtes the size of the column is bigger than the size of the control. In this case the "align" property specify the position of the control inside the column.	Optional	top middle bottom