File Upload Handling

When an application is required to receive input as a file (txt, csv or excel for example) to process it's contents, we can use the following steps to include a FileUploadHandler on the application.

The FileUploadHandler will be responsible for receiving a file from the client, and store it on a server folder for further processing. Using an handler such as this, means that we're clearly separating the logic that uploads the file and stores it to disk and the moment where you actually process that file (which usually happens in the action trigger of a button pressed by the user to instruct the processing of the uploaded file).

Step-by-step guide

First of all lets start with the steps that have to be done on the server side:

Create support for handling the upload

C#

1. Start by creating a FileUploadHandler.ashx file with the following markup:

FileDownloadHandler.ashx

```
<%@ WebHandler Language="C#"
class="AppSupportLib.Web.Services.Runtime.Web.FileUploadHandler" %>
```

Note:
It's possible to use a custom upload handler (instead of the one present on the Foundations Framework), by creating a class that implements the IHttpHandler interface and replace the class attribute value on the file above with the created class full name.

JAVA

1. Create support on the servlet file - web.xml

web.xml

```
<servlet>
    <servlet-name>upload</servlet-name>

<servlet-class>morphis.foundations.core.appsupportlib.runtime.web.FileUplo
adServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>upload</servlet-name>
    <url-pattern>/upload/*</url-pattern>
  </servlet-mapping>
  <servlet>
```

Note:
It's possible to use a custom upload handler (instead of the one present on the Foundations Framework), by creating a class that implements the HttpServlet Class and replace the class attribute value on the file above with the created class full name.

Configure where the upload are placed

C#

1. Open the web.config file, and correctly configure the folder where the uploaded files will be placed:

   web.config upload path configuration

   ```
   <appSupportLibSettings
     ...
        uploadPath="~/uploadDropZone/" />
   ```

   Note:
   If using the upload handler present on the Foundations Framework, the developer needs to create specified folder on the server in order for the handler to correctly save the file. The handler doesn't create server folders if they are not found!

2. Open the config.xml file and set the UPLOAD_URL global parameter with the FileUploadHandler.ashx file path:

   config.xml upload_url parameter configuration

   ```
   <param name="UPLOAD_URL" value="/<server_path>/FileUploadHandler.ashx"
   />
   ```

3. Prepare the server-side action to process the uploaded file. Here's an example of the code you could use to either process a single file or a zip with multiple files to process:

ProcessUploadedFile method

```
public virtual void ProcessUploadedFile()
{
 try
 {
  String uploadedDropZone =
Foundations.Core.AppSupportLib.Configuration.AppSupportLibSettings.Ge
tConfig().UploadPath;
  String uploadedFile =
System.Web.HttpContext.Current.Request.MapPath(uploadedDropZone) +
Model.BlockName.ItemName;

  if (File.Exists(uploadedFile))
  {
   IList<string> files2Process = null;

   if (uploadedFile.ToLower().EndsWith(".zip"))
    files2Process = UnzipFile(uploadedFile);
   else
    files2Process = new List<string>() { uploadedFile };

   foreach (string file in files2Process)
   {
    // process the file's contents
   }
  }
 }
}
```

Note:
This code assumes that if the user uploads a zip file, it's contents should be unzipped into a list of files to process.

JAVA

1. Open the appsupportlib.config.xml and correctly configure the folder where the uploaded files will be placed:

appsupport.config.xml

```
<bean id="appSupportLibSettings"

class="morphis.foundations.core.appsupportlib.configuration.AppSuppor
tLibSettings"
        init-method="initialize">

        <property name="envVars">
            <map>
                <entry key="uploadPath">
                    <value>/temp</value>
                </entry>
```

Note:
If using the upload handler present on the Foundations Framework, the developer needs to create specified folder on the server in order for the handler to correctly save the file. The handler doesn't create server folders if they are not found!

2. Open the appsupportlib.config.xml and set the fileDownloadProcessor global parameter with the FileUploadHandler you wish to use, that must implement IFileUploadProcessor Interface:

appsupportlib.config.xml

```
<property name="fileUploadProcessor">
            <bean
class="morphis.foundations.core.appsupportlib.runtime.web.actions.upl
oad.DefaultFileUploadProcessor"/>
        </property>
```

Note:
If none is configured the default of the Foundations Framework will be used, the DefaultFileUploadProcessor Class

3. Prepare the server-side action to process the uploaded file. Here's an example of the code you could use to either process a single file and insert on a Blob:

ProcessUploadedFile method

```
public virtual void ProcessUploadedFile()
{
  //Row that contains a blob
   DownloadAdapter downloadElement = (DownloadAdapter)
this.getFormModel().getDownload().getRowAdapter(true);
        // upload file to virtual field and update the inserted row in
post
        // insert
        NString key = downloadElement.getFileNameSelect();
        try {

            NString filePath = FileManager.getUploadedFilePath(key);
            NBlob fichData = FileManager.readUploadedFile(filePath);
            if (fichData.isNull()) {
                errorMessage("*ERROR* File must exist and be selected
to upload");
                throw new ApplicationException();
            }
   //Insert the file in the virtual field that is a NBlob
            downloadElement.setDFile(fichData);

        } catch (IOException e) {
            errorMessage("An error occurred during the upload
process.");
            throw new ApplicationException();
        }
}
```

Client Side

Lets move now to the steps that have to be done on the client side:

1. Place the filebox widget on the .xvc file. This widget will allow the user to select a file from his computer to upload to the server:

xvc file

```
<filebox label="Item Name" labelposition="left" labeloffset="5"
name="itemName" left="0px" top="0px" width="100px" height="19px"
member="ITEM_NAME" block="BLOCK_NAME" maxlength="100"
datatype="String" />
```

Note:
This widget doesn't allow the user to select multiple files for upload, but there's the possibility to upload a zip file (containing all the files the user wishes to upload) and then create code (on the server side) to extract the contents of the zip file and process them one by one.

Also note, that this widget implies that you've also created an item named "ITEM_NAME" in block "BLOCK_NAME".

Related articles

Error rendering macro 'contentbylabel' : parameters should not be empty