# 파일 업로더 기본 가이드

Last update: 2024/07/15

# 목차

# 파일 업로더 기본 가이드

이 화면은 FileUpload 컴포넌트에 대한 샘플 화면이다.

FileUpload 컴포넌트는 HTTP Multipart 통신 프로토콜 표준을 따르는 파일 업로드 처리 컴포넌트이다.

관련 속성으로 with_credentials, postdata_encode가 있다.

관련 API로 seturl, addfile, addfileobjectarray, deleteallfile, clearallfilestatus, startupload, stopupload가 있다.

파일 정보 관련 API로 getfilecount, getfilename, getfilesize, getfilebriefsize, getfiledate, getfiletime이 있다.

파일 업로드 상태 관련 API로 getfilestatus, getfileprogress, getfileresult, getfileresultmsg, getfileresultfilename가 있다.

관련 이벤트로 on_listupdate, on_fileprogress, on_filecomplete가 있다.

## 예시

템플릿 위치: /HTML5/COMPONENT/FILEUPLOADER/fileuploader_basic

템플릿 파일

- fileuploader_basic.xml
- fileuploader_basic.js
- fileuploader.jsp.txt
- 새창으로 실행

### 화면 스크립트

```
// 툴 설치 디렉토리\technet\project\template\ext\java\fileuploader.jsp.txt 파일을
fileuploader.jsp 이름으로 변경후,
// 해당 파일을 WAS 서버에 적용하고, 해당 서버 URL로 아래의 base_url 변수값을 변경후 테스트
진행해야 함
var base_url = "http://127.0.0.1:8080/xframe5/fileuploader.jsp";

// 화면 로드 이벤트
function screen_on_load()
{
  // 파일 업로드 대상 URL 지정
    this.uploader_basic.seturl(base_url);
}

// "파일 추가" 버튼 이벤트
```

```
function btn_addfile_on_mouseup(objInst)
{
    this.uploader_basic.addfile();
}

// "모든 파일 삭제" 버튼 이벤트
function btn_deleteallfile_on_mouseup(objInst)
{
  // 모든 업로드 대상 파일 삭제
    this.uploader_basic.deleteallfile();
}

// "업로드 상태 초기화" 버튼 이벤트
function btn_clearallfilestatus_on_mouseup(objInst)
{
  // 모든 파일에 대한 업로드 상태 초기화
    this.uploader_basic.clearallfilestatus();

  // 업로드 상태 초기화를 그리드에 표시
    this.UpdateFileStauts(-1);
}

// "업로드 시작" 버튼 이벤트
function btn_startupload_on_mouseup(objInst)
{
    this.uploader_basic.startupload();
}

// "업로드 중지" 버튼 이벤트 처리
function btn_stopupload_on_mouseup(objInst)
{
    this.uploader_basic.stopupload();
}

// 업로드 파일 상태 정보 업데이트 처리
function UpdateFileStauts(nFileIndex) {
    var count, i;

  // 파일 인덱스가 -1인 경우, 전체 파일 목록에 대해서 업데이트 처리
    if (nFileIndex == -1) {
        count = this.uploader_basic.getfilecount();
        for (i = 0; i < count; i++) {
            this.UpdateOneFileStauts(i);
        }
    }
  // 파일 인덱스가 -1이 아닌 경우, 특정 파일에 대해서 업데이트 처리
    else {
        this.UpdateOneFileStauts(nFileIndex);
    }
}

// 업로드 대상 파일 목록 정보를 그리드에 다시 표시
function ReloadFileStatus() {
    var nFileIndex, count;

   // 그리드 내용 전체 지움
    this.grdList.deleteall();

   // 업로드 대상 파일 갯수만큼 Loop를 돌면서 그리드에 추가
    count = this.uploader_basic.getfilecount();
    for (nFileIndex = 0; nFileIndex < count; nFileIndex++) {
        this.grdList.additem(false, false);

        this.grdList.setitemtextex(nFileIndex, 0,
this.uploader_basic.getfilename(nFileIndex), false);
        this.grdList.setitemtextex(nFileIndex, 1,
this.uploader_basic.getfilesize(nFileIndex), false);
        this.grdList.setitemtextex(nFileIndex, 2,
this.uploader_basic.getfilebriefsize(nFileIndex), false);
        this.grdList.setitemtextex(nFileIndex, 3,
this.uploader_basic.getfiledate(nFileIndex), false);
        this.grdList.setitemtextex(nFileIndex, 4,
this.uploader_basic.getfiletime(nFileIndex), false);

        this.UpdateOneFileStauts(nFileIndex, false);
    }

    this.grdList.refresh();
}

// 한 파일에 대한 업로드 상태 표시 업데이트
function UpdateOneFileStauts(nFileIndex, bRefresh) {
    this.grdList.setitemtextex(nFileIndex, 5,
this.uploader_basic.getfilestatus(nFileIndex), bRefresh);
    this.grdList.setitemtextex(nFileIndex, 6,
this.uploader_basic.getfileprogress(nFileIndex), bRefresh);
    this.grdList.setitemtextex(nFileIndex, 7,
this.uploader_basic.getfileresult(nFileIndex), bRefresh);
    this.grdList.setitemtextex(nFileIndex, 8,
this.uploader_basic.getfileresultmsg(nFileIndex), bRefresh);
    this.grdList.setitemtextex(nFileIndex, 9,
this.uploader_basic.getfileresultfilename(nFileIndex), bRefresh);
}

////////////////////////////////////////////////////////////////////////////
/////////////////
// EVENT
////////////////////////////////////////////////////////////////////////////
/////////////////

// 그리드 파일 드롭 이벤트 처리 (탐색시에서 파일 Drag&Drop 처리시 발생함)
function grdList_on_dropfiles(objInst, arrayDropFiles, nDropFileCount)
{
    var     i, fileObj;
   // 드롭된 파일 갯수 및 파일 정보를 콘솔에 출력
     factory.consoleprint("nDropFileCount = " + nDropFileCount);
     for (i = 0; i < nDropFileCount; i++) {
         fileObj = arrayDropFiles[i];
         factory.consoleprint(i + " : fileObj.name = " + fileObj.name);
         factory.consoleprint(i + " : fileObj.size = " + fileObj.size);
     }

   // 드롭된 파일 오브젝트 배열을 업로드 대상에 추가
     this.uploader_basic.addfileobjectarray(arrayDropFiles);
}

// 파일 업로드 컴포넌트 개별 파일 업로드 진행 상태 이벤트 처리
function uploader_basic_on_fileprogress(objInst, nFileIndex, strFileName,
nPos)
{
   // 파일 업로드 진행 상태 업데이트
     this.grdList.setitemtext(nFileIndex, 6, nPos);
}

// 파일 업로드 컴포넌트 개별 파일 럽로드 완료 이벤트 처리
function uploader_basic_on_filecomplete(objInst, nFileIndex, strFileName)
{
   // 파일 업로드 완료 상태 업데이트
     this.UpdateOneFileStauts(nFileIndex);
}

// 파일 업로드 컴포넌트 업로드 대상 목록 변경 이벤트 처리
function uploader_basic_on_listupdate(objInst)
{
   // 업로드 대상 파일 목록 정보를 그리드에 다시 표시
     this.ReloadFileStatus();
}
```

## fileuploader.jsp 소스

```
<%@ page import="java.io.File" %>
<%@ page import="java.io.IOException" %>
<%@ page import="java.io.PrintWriter" %>
<%@ page import="java.io.UnsupportedEncodingException" %>
<%@ page import="java.util.HashMap" %>
<%@ page import="java.util.Iterator" %>
<%@ page import="java.util.List" %>

<%@ page import="org.apache.commons.fileupload.FileItem" %>
<%@ page import="org.apache.commons.fileupload.FileUploadException" %>
<%@ page import="org.apache.commons.fileupload.disk.DiskFileItemFactory" %>
<%@ page import="org.apache.commons.fileupload.servlet.ServletFileUpload" %>
<%@ page import="org.apache.commons.io.FilenameUtils" %>
<%@ page import="org.apache.log4j.Logger" %>

<%!
    // create logging object
    Logger     logger = Logger.getLogger(getClass());

    // Define WebFileManager(WFM) Constant
    String    WFM_DATA_DEL        = String.valueOf((char)0x1A);        //
data delimeter
    String    WFM_DATASTART_DEL    = String.valueOf((char)0x1C);        //
data start indicator��
    String    WFM_DATAEND_DEL        = String.valueOf((char)0x1F);        //
data end indicator��
    String    WFM_SUCCESS            = "success";                        //
success message
    String    WFM_ERROR            = "error";                          //
error message
    String    WFM_SAVE_FILE_NAME    = "SaveFileName";                   //
saved file name parameter key

    int        maxMemoryFileSize = 10;                // maximum memory file
size
    int        maxFileSize =  1000 * 1024 * 1024;    // maximum file size
(100MB)

    String    errorMsg = "";
    String    contextRootDir = "";
    String    tempDirAbsolutePath = "";
    String    saveBaseDirAbsolutePath = "";
%>

<%
    ServletFileUpload     uplaodHandler = null;            // file uplaod
handler
    List                 fileItemList = null;            // file item list
    FileItem              fileItem = null;               // file item
    HashMap                 paramMap = new HashMap();        // parameter map

logger.info("=============================================================="
);
    logger.info("file upload start");

    // Set cross domain response header
    /*
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Headers", "X-Requested-With");
    */

    // Reference: XDataSet5.jar
    response.setHeader("Access-Control-Allow-Credentials", "true");
    if(request == null) {
        logger.error("Access-Control-Allow-Origin = *");
        response.setHeader("Access-Control-Allow-Origin", "*");
    }
    else {
        logger.error("Access-Control-Allow-Origin = " +
request.getHeader("Origin"));
        response.setHeader("Access-Control-Allow-Origin",
request.getHeader("Origin"));
    }
    response.setHeader("Access-Control-Allow-Headers", "X-Requested-With");


    // Set upload file temp dir, save base dir path
    // setUploadEnvSetting(getServletContext());        // servlet 3.0 spec
    setUploadEnvSetting(request.getSession().getServletContext());

    // Check whether the request has multipart data content.
    boolean isMultipart = ServletFileUpload.isMultipartContent(request);
    if(!isMultipart) {
        try {
            logger.error("There is no multipart data in request");
            out.print(getErrorMsg("There is no multipart data in request"));
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        return;
    }

    // Create a file upload handler
    uplaodHandler = getFileUploadProcessor();
    logger.info("Success to create a file upload handler");

    // parse the request
    try {
        fileItemList = uplaodHandler.parseRequest(request);
    }
    catch(FileUploadException ex) {
        try {
            logger.error("Fail to parse a request");
            out.print(getErrorMsg("Fail to parse a request"));
        }
        catch (UnsupportedEncodingException e) {
            logger.error("Exception msg = " + e.getMessage());
            e.printStackTrace();
        }
        return;
    }

    logger.info("Success to parse a request");

    // Process the parameter
    Iterator iter1 = fileItemList.iterator();
    while (iter1.hasNext()) {
        fileItem = (FileItem)iter1.next();

        // process a regular form field
        if (fileItem.isFormField()) {
            String fieldName = fileItem.getFieldName();
            String fieldValue = fileItem.getString();
            logger.info("fieldName = " + fieldName + ", fieldValue = " +
fieldValue);
            paramMap.put(fieldName, fieldValue);
        }
    }

    Iterator iter2 = fileItemList.iterator();
    while (iter2.hasNext()) {
        fileItem = (FileItem)iter2.next();

        // process a regular form field
        if (fileItem.isFormField()) {
            continue;
        }

           try {
               String returnMessage = null;

               // process a upload multipart data
               String saveFileName = handleUploadFile(fileItem, paramMap);
               if(saveFileName == null) {
                   returnMessage = getErrorMsg(errorMsg);
               }
               else {
                // make a success message
                   returnMessage = getSuccessMsg(saveFileName);
               }

               // return a success message to client
            logger.info("returnMessage = [" + returnMessage + "]");

            // out.clearBuffer();
               out.print(returnMessage);
           }
           catch(Exception ex) {
            logger.error("Exception Msg = " + ex.getMessage());
            ex.printStackTrace();
            out.print(getErrorMsg("Fail to process upload file."));
           }
    }
%>

<%!
```

```
// Set upload file temp dir, save base dir path
public void setUploadEnvSetting(ServletContext context)
{
    // get context real path
    contextRootDir = context.getRealPath("/");
    if(contextRootDir.endsWith(File.separator) == false) {
        contextRootDir += File.separator;
    }

    // temporary directory absolute path for temporary file
    tempDirAbsolutePath = contextRootDir + "temp";
    saveBaseDirAbsolutePath = contextRootDir + "upload";

    logger.info("tempDirAbsolutePath = " + tempDirAbsolutePath);
    logger.info("saveBaseDirAbsolutePath = " + saveBaseDirAbsolutePath);

    makeDirUsingDirPath(tempDirAbsolutePath);
    makeDirUsingDirPath(saveBaseDirAbsolutePath);

    return;
}

// handle a upload file data
public String handleUploadFile(FileItem fileItem, HashMap paramMap) throws
Exception
{
    String        saveFileAbsolutePath = "";        // file absolute path to
save
    String        saveFileName = "";

    String         filePath = fileItem.getName();                // HTML의
File 형식의 입력 콘트롤을 이용하여 지정한 로컬 파일 경로
    String        fileName = FilenameUtils.getName(filePath);    // file
name

    String         paramDirPath = "";
    String         paramFileName = "";

    logger.info("filePath = " + filePath);
    logger.info("fileName = " + fileName);
    logger.info("getContentType = " + fileItem.getContentType());
    logger.info("getSize = " + fileItem.getSize());

    // get file save information
    paramDirPath = paramMap.get("DIR_PATH") == null ? "" :
(String)paramMap.get("DIR_PATH");
    paramFileName = paramMap.get("FILE_NAME") == null ? "" :
(String)paramMap.get("FILE_NAME");

    logger.info("paramDirPath = [" + paramDirPath + "]");
    logger.info("paramFileName = [" + paramFileName + "]");
    // set save directory absolute path
    saveFileAbsolutePath = saveBaseDirAbsolutePath;
    if(paramDirPath.length() > 0) {
        saveFileAbsolutePath = saveFileAbsolutePath + File.separatorChar +
paramDirPath;
    }

    // set save file absolute path
    if(paramFileName.length() > 0) {
        saveFileName = paramFileName;
    }
    else {
        saveFileName = fileName;
    }
    saveFileAbsolutePath = saveFileAbsolutePath + File.separator +
saveFileName;

    logger.info("saveFileAbsolutePath = " + saveFileAbsolutePath);

    // make a directory for file path
    makeDirUsingFilePath(saveFileAbsolutePath);

    // save a upload file to a save file absolute path
    while(true) {
        int    retryCount = 0;

        try {
            fileItem.write(new File(saveFileAbsolutePath));
            break;
        }
        catch(Exception e) {
            logger.error("Exception Msg = " + e.getMessage());
            retryCount++;

            if(retryCount > 5) {
                logger.error("Fail to wirte a file");
                errorMsg = "Fail to wirte a file";
                return null;
            }
            else {
                try {
                    Thread.sleep(1000);
                }
                catch (InterruptedException ignore) {
                    ;
                }
                continue;
            }
        }
    }
    logger.info("Success To Wirte File");

    // delete a file item content�
    fileItem.delete();

    // return a saved file name
    return saveFileName;
}

//TODO: change a character set of message
public String msgCharacterSetConvert(String message) throws
UnsupportedEncodingException
{
    if(message == null) {
        return "";
    }

    /*
    logger.info("don't change character set");
    return message;
    */

    logger.info("change character set UTF-8 -> ISO-8859-1");
    return new String(message.getBytes("UTF-8"), "ISO-8859-1");
}

// make a success message
public String getSuccessMsg(String saveFileName) throws
UnsupportedEncodingException {
    StringBuffer    returnMsg = new StringBuffer();

    returnMsg.append(WFM_DATASTART_DEL);
    returnMsg.append(WFM_SUCCESS);
    returnMsg.append(WFM_DATA_DEL);
    returnMsg.append(WFM_SAVE_FILE_NAME + "=" + saveFileName);
    returnMsg.append(WFM_DATAEND_DEL);

    /*
    logger.info("String.valueOf((char)0x1A) = " +
String.valueOf((char)0x1A));
    logger.info("String.valueOf((char)0x1C) = " +
String.valueOf((char)0x1C));
    logger.info("String.valueOf((char)0x1F) = " +
String.valueOf((char)0x1F));
    logger.info("WFM_DATASTART_DEL = [" + WFM_DATASTART_DEL+ "]");
    logger.info("WFM_DATA_DEL = [" + WFM_DATA_DEL + "]");
    logger.info("WFM_DATAEND_DEL = [" + WFM_DATAEND_DEL + "]");

    logger.info("returnMsg = [" + returnMsg.toString() + "]");
    */
    return msgCharacterSetConvert(returnMsg.toString());
}

// make a error message
public String getErrorMsg(String errorMsg) throws
UnsupportedEncodingException
{
    StringBuffer    returnMsg = new StringBuffer();

    returnMsg.append(WFM_DATASTART_DEL);
    returnMsg.append(WFM_ERROR);
    returnMsg.append(WFM_DATA_DEL);
    returnMsg.append(errorMsg);
    returnMsg.append(WFM_DATAEND_DEL);

    return msgCharacterSetConvert(returnMsg.toString());
}

private String getRandomFileName() {
    return java.util.UUID.randomUUID().toString().replace("-", "");
}

// get a file upload process
private ServletFileUpload getFileUploadProcessor()
{
    // create a new file item factory
    DiskFileItemFactory factory = new DiskFileItemFactory();

    // create a new file object for a temporary directory
    File    tempDir = new File(tempDirAbsolutePath);

    // create a temporary directory
    if(!tempDir.exists()) {
        tempDir.mkdirs();
    }

    // set a temporary directory
    factory.setRepository(tempDir);

    // set a maximum memory file size
    factory.setSizeThreshold(maxMemoryFileSize);

    // create a servlet file upload object using a factory
    ServletFileUpload upload = new ServletFileUpload(factory);

    // set a maixmum file size
    upload.setSizeMax(maxFileSize);

    // set header encoding characterset for hangul file name
    upload.setHeaderEncoding("UTF-8");
    return upload;
}

// make a directory using a file path
public void makeDirUsingFilePath(String fileAbsolutePath)
{
    File     oFile = new File(fileAbsolutePath);
    File    oDir = oFile.getParentFile();

    // create a diectory
    if(!oDir.exists()) {
        oDir.mkdirs();
    }
}

// make a directory using a dir path
public void makeDirUsingDirPath(String dirAbsolutePath)
{
    File     oDir = new File(dirAbsolutePath);

    // create a diectory
    if(!oDir.exists()) {
        oDir.mkdirs();
    }
}
%>
```