Turbine2 Tutorial DirectoryService

Integrating FCKEditor into Turbine

FCKEditor (http://www.fckeditor.net) is a great Javascript based editor. It can manage server repositories to store files for documents. Let us make a server side interface for this. Planning considerations:

- there should be a lot of subrepository to store its data in server, for example, user´s introductions, news parts, etc.
- All of entity can make subdirectories into own repository. It´s optional controlling by TR.properties
- All of entity´s subdirectory can has a special directory named ""private"" to serve data for logged in users.
- All of subrepository has quotas to reduce size of entity store.
- File serving is made by Turbine to controlling access to files in repositories.

Making TurbineDirectory Service

there are 3 class of this package

DirectoryService.java

It is an interface for constants and method declarations.

```
package com.zamek.portal_zamek.services.directory;

import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.List;

import org.apache.commons.fileupload.FileItem;
import org.apache.turbine.services.Service;

import com.zamek.portal_zamek.PortalException;
import com.zamek.portal_zamek.util.FileInfo;

public interface DirectoryService extends Service {

    /**
     * The key under which this service is stored in TurbineServices.
     */
    String SERVICE_NAME = "DirectoryService";

    /**
     * repository part in url
     */
    String CTX_REPO = "repo";

    /**
     * id part in url
     */
    String CTX_ID = "id";

    /**
     * file part in url
     */
    String CTX_FILE = "file";

    /**
     * FileServ action part of link
     */
    String CTX_FILE_SERV = "FileServ";
    /**
     * repository path key in config
     */
    String REPOSITORY_KEY = "repository";

    /**
     * default base path
     */
    String DEFAULT_REPOSITORY = "resources";
```

```
/**
 * default enable private
 */
boolean DEFAULT_PRIVATE = false;

/**
 * quota key in config
 */
String QUOTA_KEY = "quota";

/**
 * Private directory enable key in config
 */
String KEY_PRIVATE_ENABLED = "private.enabled";

/**
 * create directory enabled key in config
 */
String KEY_CREATE_DIRECTORY_ENABLED = "create-directory.enabled";

/**
 * quota key in config
 */
String KEY_QUOTA = "quota";
/**
 * default quota of each directory size
 */
int DEFAULT_QUOTA = 1024000;

/**
 * index of resourcebundle to quota exceeded
 */
String ERR_QUOTA_EXCEEDED = "quota_exceede";

String ERR_CANNOT_OPEN_FILE = "cannot_open_file";

String MASK_PRIVATE = "/private";
/**
 * Folder already exists error for FCKEditor
 */
public final static int ERR_FOLDER_EXISTS =     101;

/**
 * Invalid folder name error for FCKEditor
 */
public final static int ERR_INVALID_FOLDER = 102;

/**
 * You have no permissions to create the folder error for FCKEDitor
 */
public final static int ERR_NO_PERMISSION = 103;

/**
 * Unknown error creating folder for FCKEditor
 */
public final static int ERR_UNKNOWN = 110;


/**
 * getting list of folders in given folder for FCKEditor.
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id  entity id, root of entity
 * @param path url in own directory
 * @param includeMask  default mask is *, if it's null
 * @param excludeMask    default mask is null
 * @return list of directories in url directoy
 * @throws FileNotFoundException
 */
public List<String> getFolders (final String subRepo, final String id,
            final String url, final String includeMask, final String excludeMask)
```

```
            throws FileNotFoundException;

    /**
     * getting list of files in given folder
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param url path url in own directory
     * @param includeMask includeMask  default mask is *, if it's null
     * @param excludeMask excludeMask    default mask is null
     * @return list files in url directory
     * @throws FileNotFoundException
     */
    public List<FileInfo> getFiles(final String subRepo, final String id,
                final String url, final String includeMask, final String excludeMask)
                throws FileNotFoundException;

    /**
     * upload file to own directory using its original name
     * You can set quota in TR.properties in
     *     services.directoryservice.<subrepo>.quota=nn
     * quota is unlimited if it's 0
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param path url in own directory
     * @param fileItem file to be upload
     * @return true if success
     * @throws PortalException
     * @throws Exception
     */
    public boolean uploadItemFile (final String subRepo, final String id,
                final String path, final FileItem fileItem)
                throws PortalException, Exception ;

    /**
     * upload file to own directory using newName
     * You can set quota in TR.properties in
     *     services.directoryservice.<subrepo>.quota=nn
     * quota is unlimited if it's 0
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param path url in own directory
     * @param fileItem file to be upload
     * @param newName rename to this new name
     * @return true if success
     * @throws PortalException
     * @throws Exception
     */
    public boolean uploadItemFile (final String subRepo, final String id,
                final String path, final FileItem fileItem, final String newName)
                throws PortalException, Exception;

    /**
     * checking to has private directory
     * in TR.properties you can enable private directory:
     * services.directoryservice.<subrepo>.private.enabled=true
     * private directory is readable only for users, not to anonymous
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @return if private.enabled is true
     */
    public boolean hasPrivateDir (final String subRepo, final String id);

    /**
     * get a file by url from repository
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param fileName name of file
     * @return bytes of file
     * @throws FileNotFoundException
     * @throws IOException
     * @throws PortalException
```

```
 */
public ByteArrayOutputStream getFile (final String subRepo, final String id, final String fileName)
            throws FileNotFoundException, IOException, PortalException;

/**
 * create a directory in repository if it's enabled by TR.properties
 * services.directoryservice.<subrepo>.create-directory.enabled
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param folder folder to create
 * @return
 *               0: OK
 *             101: Folder already exists error for FCKEditor
 *                102: Invalid folder name error for FCKEditor
 *             103: You have no permissions to create the folder error for FCKEDitor
 *             110: Unknown error creating folder for FCKEditor
 * @throws FileNotFoundException
 */
public int createDirectory (final String subRepo, final String id, final String folder)
            throws FileNotFoundException;

/**
 * save string to repository to given name
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
 * @param fileName name of file to save
 * @param data data to save
 * @throws IOException
 */
public void saveToFile(final String subRepo, final String id,
            final String path, final String fileName, final String data)
            throws IOException, PortalException;

/**
 * checking given file to exists
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
 * @param fileName name of file to check
 * @return true if exists
 * @throws FileNotFoundException
 */
public boolean isFileExists (final String subRepo, final String id,
            final String path, final String fileName)
            throws FileNotFoundException;

/**
 * create directory of identity in subRepo
 */
public boolean createIdDirectory (final String subRepo, final String id)
    throws FileNotFoundException;

/**
 * getting quota of subRepo
 * @param subRepo
 * @return 0 if no quota
 */
public long getQuota (final String subRepo);

/**
 * getting real path of subrepo
 * @param subRepo
 * @return
 */
public String getBasePath(final String subRepo);

/**
 * getting enable create directory property
 * @param subRepo
 * @return
```

```
     */
    public boolean getCreateDirEnabled(final String subRepo);

    /**
     * getting enable private directory property
     * @param subRepo
     * @return
     */
    public boolean getPrivateDirEnabled (final String subRepo);

    /**
     * checking subrepo exists in TR.properties
     * @param subRepo
     * @return
     */
    public boolean isSubRepoEnabled(final String subRepo);

    /**
     * create a link for a file
     * @param subRepo
     * @param id
     * @param fileName
     * @return
     */
    public String getFileUrl(final String subRepo, final String id, final String fileName) ;
}
```

TurbineDirectoryService.java

This is the core of '***T***urbine'***D***irectory service.

```
package com.zamek.portal_zamek.services.directory;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;

import org.apache.commons.configuration.Configuration;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.turbine.Turbine;
import org.apache.turbine.services.InitializationException;
import org.apache.turbine.services.TurbineBaseService;
import org.apache.turbine.services.localization.Localization;
import org.apache.turbine.util.uri.TemplateURI;

import com.zamek.portal_zamek.PortalException;
import com.zamek.portal_zamek.util.FileInfo;
import com.zamek.portal_zamek.util.FileUtils;

public class TurbineDirectoryService extends TurbineBaseService implements
        DirectoryService {

    private static Log log = LogFactory.getLog(TurbineDirectoryService.class);

    private final static int BUFFER_SIZE = 0x4000;

    private String repoPath = null;

    /*
     * list of disabled directory names
     */
```

```
enum DisabledFolderNames
{
    ROOT ("^/");

    private final String item;

    DisabledFolderNames(String item) { this.item = item; }
    String getItem () { return this.item; }
}


public TurbineDirectoryService() {

}

/**
 * initialisation. Get repository path from TR.prop
 */
public void init () throws InitializationException
{
    repoPath = getConfiguration().getString(
            DirectoryService.REPOSITORY_KEY,
            DirectoryService.DEFAULT_REPOSITORY);

    String testPath = Turbine.getRealPath(repoPath);
    File testDir = new File(testPath);
    if (!testDir.exists())
    {
        if (!testDir.mkdirs())
            throw new InitializationException("Couldn't create target directory!");

    }
    repoPath = testPath;
    getConfiguration().setProperty(DirectoryService.REPOSITORY_KEY, repoPath);
    log.debug("DirectoryService path is:" + repoPath);
    setInit(true);

}

/**
 * checking subrepo exists in TR.properties
 * @param subRepo
 * @return
 */
public boolean isSubRepoEnabled(final String subRepo)
{
    return StringUtils.isNumeric(getConfiguration().getString(subRepo + '.' + DirectoryService.KEY_QUOTA));
}

/**
 * getting list of folders in given folder for FCKEditor.
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id  entity id, root of entity
 * @param path url in own directory
 * @param includeMask  default mask is *, if it's null
 * @param excludeMask    default mask is null
 * @return list of directories in url directoy
 * @throws FileNotFoundException
 */
public List<String> getFolders (final String subRepo, final String id, final String url,
            final String includeMask, final String excludeMask)
            throws FileNotFoundException
{
    String basePath = getBasePath(subRepo, id);

    return basePath != null
            ? FileUtils.getDirectoryNames(basePath+File.separatorChar +
                    securityCheck (url), includeMask, excludeMask)
            : null;
}
```

```
/**
 * getting list of files in given folder for FCKEditor
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param url path url in own directory
 * @param includeMask includeMask  default mask is *, if it's null
 * @param excludeMask excludeMask    default mask is null
 * @return list files in url directory
 * @throws FileNotFoundException
 */
public List<FileInfo> getFiles(final String subRepo, final String id, final String url,
            final String includeMask, final String excludeMask)
            throws FileNotFoundException
{
    String basePath = getBasePath(subRepo, id);
    return basePath != null
            ? FileUtils.getFileNames(basePath+File.separatorChar +
                    securityCheck (url), includeMask, excludeMask)
            : null;
}

/**
 * upload file to own directory using its original name
 * You can set quota in TR.properties in
 *     services.directoryservice.<subrepo>.quota=nn
 * quota is unlimited if it's 0
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param path url in own directory
 * @param fileItem file to be upload
 * @return true if success
 * @throws PortalException
 * @throws Exception
 */
public boolean uploadItemFile (final String subRepo, final String id,
            final String path, final FileItem fileItem)
            throws PortalException, Exception
{
    return uploadItemFile(subRepo, id, path, fileItem, fileItem.getName());
}

/**
 * upload file to own directory using newName
 * You can set quota in TR.properties in
 *     services.directoryservice.<subrepo>.quota=nn
 * quota is unlimited if it's 0
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param path url in own directory
 * @param fileItem file to be upload
 * @param newName rename to this new name
 * @return true if success
 * @throws PortalException
 * @throws Exception
 */
public boolean uploadItemFile (final String subRepo, final String id,
            final String path, final FileItem fileItem, final String newName)
            throws PortalException, Exception
{
    String basePath = getBasePath(subRepo, id);
    if (basePath == null)
        return false;
    Configuration conf = getConfiguration();
    int quota = conf.getInt(subRepo + '.' + DirectoryService.KEY_QUOTA, 0);
    // checking quotas
    if (quota > 0 &&
            ((getSize(basePath) + fileItem.getSize()) > quota))
        throw new PortalException (Localization.getString(ERR_QUOTA_EXCEEDED));

    try
    {
```

```
        FileUtils.uploadFile(fileItem, basePath+securityCheck (path), securityCheck(newName));
        return true;
    }
    catch (Exception e) {
        return false;
    }
}

/**
 * checking to has private directory
 * in TR.properties you can enable private directory:
 * services.directoryservice.<subrepo>.private.enabled=true
 * private directory is readable only for users, not to anonymous
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @return if private.enabled is true
 */
public boolean hasPrivateDir (final String subRepo, final String id)
{
    return getConfiguration().getBoolean(subRepo + '.' + DirectoryService.KEY_PRIVATE_ENABLED);
}

/**
 * get a file by url from repository
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param fileName name of file
 * @return bytes of file
 * @throws FileNotFoundException
 * @throws IOException
 * @throws PortalException
 */
public ByteArrayOutputStream getFile (final String subRepo, final String id,
            final String fileName)
            throws FileNotFoundException, IOException, PortalException
{
    //getting basePath
    String basePath = getBasePath(subRepo, id);
    if (basePath == null)
        return null;
    File file = new File (basePath + File.separatorChar + securityCheck(fileName));
    if (file.isFile() && file.exists() && file.canRead())
    {
        FileInputStream fis = new FileInputStream(file);
        ByteArrayOutputStream bos = new ByteArrayOutputStream();

        byte[] buffer = new byte[BUFFER_SIZE];
        int len = 0;
        for(len = fis.read(buffer, 0, BUFFER_SIZE);len != -1;len = fis.read(buffer, 0 , BUFFER_SIZE))
            bos.write(buffer, 0, len);
        return bos;
    }
    else
        throw new PortalException (String.format(Localization.getString(ERR_CANNOT_OPEN_FILE),fileName));
}

/**
 * create a directory in repository if it's enabled by TR.properties
 * services.directoryservice.<subrepo>.create-directory.enabled
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param folder folder to create
 * @return
 *               0: OK
 *             101: Folder already exists error for FCKEditor
 *                102: Invalid folder name error for FCKEditor
 *             103: You have no permissions to create the folder error for FCKEDitor
 *             110: Unknown error creating folder for FCKEditor
 * @throws FileNotFoundException
 */
public int createDirectory (final String subRepo, final String id, final String folder)
```

```
        throws FileNotFoundException
{
    Configuration conf = getConfiguration();
    if (conf.getBoolean(subRepo+"."+DirectoryService.KEY_CREATE_DIRECTORY_ENABLED))
    {

        String basePath = getBasePath(subRepo, id);

        if (basePath == null || folder == null || folder.length() <= 0)
            return DirectoryService.ERR_UNKNOWN;

        // checking disabled folder names
        for (DisabledFolderNames dfn : DisabledFolderNames.values() )
            if (folder.matches(dfn.getItem()))
                    return ERR_INVALID_FOLDER;

        try
        {
            File newDir = new File (basePath + securityCheck(folder));
            if (newDir == null)
                return ERR_UNKNOWN;

            if (newDir.exists())
                return ERR_FOLDER_EXISTS;

            return newDir.mkdir() ? 0 : ERR_UNKNOWN;
        }
        catch(Exception e) {
            return ERR_INVALID_FOLDER;
        }
    }
    else
        return ERR_INVALID_FOLDER;
}

/**
 * save string to repository to given name
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
 * @param fileName name of file to save
 * @param data data to save
 * @throws IOException
 */
public void saveToFile(final String subRepo, final String id,
            final String path, final String fileName, final String data)
            throws IOException, PortalException
{
    String basePath = getBasePath(subRepo, id);

    if (basePath == null || fileName == null || fileName.length() == 0)
        return;

    int quota = getConfiguration().getInt(subRepo + '.' + DirectoryService.KEY_QUOTA, 0);

    // checking quotas
    if (quota > 0 &&
            ((getSize(basePath) + data.length()) > quota))
        throw new PortalException (Localization.getString(ERR_QUOTA_EXCEEDED));

    File file = new File (basePath + securityCheck(path) + File.separatorChar + securityCheck (fileName));
    PrintWriter out = new PrintWriter(new FileWriter(file));
    out.print(data);
    out.close();
}

/**
 * checking given file to exists
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
```

```
 * @param fileName name of file to check
 * @return true if exists
 * @throws FileNotFoundException
 */
public boolean isFileExists (final String subRepo, final String id,
            final String path, final String fileName)
            throws FileNotFoundException
{
    String basePath = getBasePath(subRepo, id);
    if (basePath == null)
        return false;

    File file = new File (basePath + securityCheck(path) + File.separatorChar + fileName);
    return file.exists();
}

public long getSize(final String basePath)
{
    return FileUtils.getSize(basePath);
}

/**
 * create directory of identity in subRepo
 */
public boolean createIdDirectory (final String subRepo, final String id)
    throws FileNotFoundException
{
    return getBasePath(subRepo, id, true) != null;
}


private String getBasePath(final String subRepo, final String id, boolean createIfNotExists)
{
    String result = new StringBuffer(repoPath).append(File.separatorChar).append(subRepo).
    append(File.separatorChar).append(id).append(File.separatorChar).toString();
    File dir = new File(result);
    if (dir.exists() && dir.isDirectory())
        return result;
    if (createIfNotExists)
        dir.mkdirs();
    return result;
}


private String getBasePath(final String subRepo, final String id) throws FileNotFoundException
{
    return getBasePath (subRepo, id, false);
}

/**
 * checking url security, removing all of ../ ./
 * @param url
 * @return
 */
private String securityCheck(final String url) {
    return url.compareTo("/") == 0
            ? ""
            : url.replaceAll("\\.+\\/", "");
}

/**
 * getting real path of subrepo
 * @param subRepo
 * @return
 */
public String getBasePath(String subRepo) {
    return new StringBuffer(repoPath).append(File.separatorChar).append(subRepo).toString();
}

/**
 * getting enable create directory property
```

```
     * @param subRepo
     * @return
     */
    public boolean getCreateDirEnabled(String subRepo) {
        return getConfiguration().getBoolean(subRepo + '.' + DirectoryService.KEY_CREATE_DIRECTORY_ENABLED);

    }

    /**
     * getting enable private directory property
     * @param subRepo
     * @return
     */
    public boolean getPrivateDirEnabled(String subRepo) {
        return getConfiguration().getBoolean(subRepo+ '.' + DirectoryService.KEY_PRIVATE_ENABLED);
    }

    /**
     * getting quota of subRepo
     * @param subRepo
     * @return 0 if no quota
     */
    public long getQuota(String subRepo) {
        return getConfiguration().getLong(subRepo+ '.' + DirectoryService.KEY_QUOTA);
    }

    public String getFileUrl(final String subRepo, final String id, final String fileName)
    {
        if (! isSubRepoEnabled(subRepo))
            return null;
        TemplateURI uri = new TemplateURI();
        uri.setScreen(CTX_FILE_SERV);
        uri.addPathInfo(CTX_REPO, subRepo);
        uri.addPathInfo(CTX_ID, id);
        uri.addPathInfo(CTX_FILE, fileName.replace('/', ','));
        return uri.toString();
    }

}
```

TurbineDirectory.java

```
package com.zamek.portal_zamek.services.directory;

import java.io.ByteArrayOutputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.List;

import org.apache.commons.fileupload.FileItem;
import org.apache.turbine.services.TurbineServices;

import com.zamek.portal_zamek.PortalException;
import com.zamek.portal_zamek.util.FileInfo;


public class TurbineDirectory {

    public static DirectoryService getService() {
        return (DirectoryService) TurbineServices.getInstance().
                getService(DirectoryService.SERVICE_NAME);
    }

    public static boolean isAvailable()
    {
        try
        {
            DirectoryService directory = getService();
            return directory != null;
```

```
    }
    catch (Exception e) {
        return false;
    }
}

/**
 * checking subrepo enabled in TR.properties
 * @param subRepo
 * @return
 */
public static boolean isSubRepoEnabled (final String subRepo)
{
    return getService().isSubRepoEnabled(subRepo);
}

/**
 * getting list of folders in given folder for FCKEditor.
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id  entity id, root of entity
 * @param path url in own directory
 * @param includeMask  default mask is *, if it's null
 * @param excludeMask    default mask is null
 * @return list of directories in url directoy
 * @throws FileNotFoundException
 */
public static List<String> getFolders (final String subRepo, final String id,
                final String url, final String includeMask, final String excludeMask)
                throws FileNotFoundException
{
    return getService().getFolders(subRepo, id, url, includeMask, excludeMask);
}

/**
 * getting list of files in given folder
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param url path url in own directory
 * @param includeMask includeMask  default mask is *, if it's null
 * @param excludeMask excludeMask    default mask is null
 * @return list files in url directory
 * @throws FileNotFoundException
 */
public static List<FileInfo> getFiles(final String subRepo, final String id,
                final String url, final String includeMask, final String excludeMask)
                throws FileNotFoundException
{
    return getService().getFiles(subRepo, id, url, includeMask, excludeMask);
}

/**
 * upload file to own directory using its original name
 * You can set quota in TR.properties in
 *     services.directoryservice.<subrepo>.quota=nn
 * quota is unlimited if it's 0
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id entity id, root of entity
 * @param path url in own directory
 * @param fileItem file to be uplode
 * @return true if success
 * @throws PortalException
 * @throws Exception
 */
public static boolean uploadItemFile (final String subRepo, final String id,
                final String path, final FileItem fileItem)
                throws PortalException, Exception
{
    return getService().uploadItemFile(subRepo, id, path, fileItem);
}

/**
```

```
     * upload file to own directory using newName
     * You can set quota in TR.properties in
     *     services.directoryservice.<subrepo>.quota=nn
     * quota is unlimited if it's 0
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param path url in own directory
     * @param fileItem file to be uplode
     * @return true if success
     * @throws PortalException
     * @throws Exception
     */
    public static boolean uploadItemFile (final String subRepo, final String id,
                    final String path, final FileItem fileItem, final String newName)
                    throws PortalException, Exception
    {
        return getService().uploadItemFile(subRepo, id, path, fileItem, newName);
    }

    /**
     * checking to has private directory
     * in TR.properties you can enable private directory:
     * services.directoryservice.<subrepo>.private.enabled=true
     * private directory is readable only for users, not to anonymous
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @return if private.enabled is true
     */
    public static boolean hasPrivateDir (final String subRepo, final String id)
    {
        return getService().hasPrivateDir(subRepo, id);
    }

    /**
     * get a file by url from repository
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param fileName name of file
     * @return bytes of file
     * @throws FileNotFoundException
     * @throws IOException
     * @throws PortalException
     */
    public static ByteArrayOutputStream getFile (final String subRepo, final String id, final String fileName)
                    throws FileNotFoundException, IOException, PortalException
    {
        return getService().getFile(subRepo, id, fileName);
    }

    /**
     * create a directory in repository if it's enabled by TR.properties
     * services.directoryservice.<subrepo>.create-directory.enabled
     * @param subRepo subsystem path in repository forexample user, news, etc.
     * @param id entity id, root of entity
     * @param folder folder to create
     * @return
     *                  0: OK
     *                101: Folder already exists error for FCKEditor
     *                   102: Invalid folder name error for FCKEditor
     *                103: You have no permissions to create the folder error for FCKEDitor
     *                110: Unknown error creating folder for FCKEditor
     * @throws FileNotFoundException
     */
    public static int createDirectory (final String subRepo, final String id, final String folder)
                    throws FileNotFoundException
    {
        return getService().createDirectory(subRepo, id, folder);
    }

    /**
     * save string to repository to given name
```

```
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
 * @param fileName name of file to save
 * @param data data to save
 * @throws IOException
 */
public static void saveToFile(final String subRepo, final String id,
                final String path, final String fileName, final String data)
                throws IOException, PortalException
{
    getService().saveToFile(subRepo, id, path, fileName, data);
}

/**
 * checking given file to exists
 * @param subRepo subsystem path in repository forexample user, news, etc.
 * @param id id entity id, root of entity
 * @param path  url in own directory
 * @param fileName name of file to check
 * @return true if exists
 * @throws FileNotFoundException
 */
public static boolean isFileExists (final String subRepo, final String id,
                final String path, final String fileName)
                throws FileNotFoundException
{
    return getService().isFileExists(subRepo, id, path, fileName);
}


public static boolean createIdDirectory(final String subRepo, final String id)
                throws FileNotFoundException
{
    return getService().createIdDirectory(subRepo, id);

}

/**
 * getting real path of subrepo
 * @param subRepo
 * @return
 */
public static String getBasePath(String subRepo)
{
    return getService().getBasePath(subRepo);
}

/**
 * getting enable create directory property
 * @param subRepo
 * @return
 */
public static boolean getCreateDirEnabled(String subRepo)
{
    return getService().getCreateDirEnabled(subRepo);
}

/**
 * getting enable private directory property
 * @param subRepo
 * @return
 */
public static boolean getPrivateDirEnabled(String subRepo)
{
    return getService().getPrivateDirEnabled(subRepo);
}

/**
 * getting quota of subRepo
 * @param subRepo
```

```
     * @return 0 if no quota
     */
    public long getQuota(String subRepo)
    {
        return getService().getQuota(subRepo);
    }

    /**
     * return a link for a file
     * @param subRepo
     * @param id
     * @param fileName
     * @return
     */
    public static String getFileUrl(final String subRepo, final String id, final String fileName)
    {
        return getService().getFileUrl(subRepo, id, fileName);
    }
}
```

setting in TR.properties

In TR.properties, you can make different settings to different subrepositories. for example in this settings has a user's repository and a news module repository

```
services.DirectoryService.classname=com.zamek.portal_zamek.services.directory.TurbineDirectoryService
services.DirectoryService.earlyInit=true
services.DirectoryService.repository=resources  # default repository root

# properties of resources/user, this is the user's own repositories
services.DirectoryService.user.quota=1024000        # 1Mb
services.DirectoryService.user.create-directory.enabled=true  # users can create subdirecories its own
repository
services.DirectoryService.user.private.enabled=true  # users can make private directory to serve files only
logged in users

# properties of news modul in resources/news
services.DirectoryService.news.quota=100000    # ~100Kb
services.DirectoryService.news.create-directory.enabled=false  #news cannott create directories
services.DirectoryService.news.private.enabled=false     #news cannot create private directory
```

Serving FCKEditor

Now we have to make an action to serve fckeditor. This is a simple '***V***'elocity'***A***'ction descendants. There are following commands from FCKEditor:

- '***G***'et'***F***'olders, need an xml response
- '***G***'et'***F***'olders'***A***'nd'***F***'iles need an xml response
- '***C***'reate'***F***'older need an xml response
- '***F***'ile'***U***'pload need a string response

In different modules you can override two of methods:

```
    boolean canCreateFolder() {
        data.getUser().hasLoggedIn() && TurbineDirectory.getCreateDirEnabled(repo);
    }
```

which is mean can create if user is has logged in and TR.properties enabled services.'***D***'irectory'***S***'ervice.<repo>.create-directory.enabled=true

```
    boolean canUpload () {
        return data.getUser().hasLoggedIn();
    }
you can override it for example return super.canUpload() && data.getAcl().hasRole(SOME_ROLE);
```

All of '***FC***'KA'*_ction is here:

```
package com.zamek.portal_zamek.modules.actions;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletResponse;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.apache.commons.fileupload.FileItem;
import org.apache.turbine.util.RunData;
import org.apache.turbine.util.parser.ParameterParser;
import org.apache.turbine.util.uri.TurbineURI;
import org.apache.velocity.context.Context;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import com.zamek.portal_zamek.ZamekConst;
import com.zamek.portal_zamek.services.directory.DirectoryService;
import com.zamek.portal_zamek.services.directory.TurbineDirectory;
import com.zamek.portal_zamek.util.FileInfo;


/**
 *
 * @author zamek
 *  request: Command=GetFolders&Type=File&CurrentFolder=/Samples/Docs/
 *  response:
 *  <?xml version="1.0" encoding="utf-8" ?>
 *    <Connector command="GetFolders" resourceType="File">
 *   <CurrentFolder path="/Samples/Docs/" url="/UserFiles/File/Samples/Docs/" />
 *   <Folders>
 *     <Folder name="Documents" />
 *     <Folder name="Files" />
 *     <Folder name="Other Files" />
 *     <Folder name="Related" />
 *   </Folders>
 * </Connector>
 *
 *  request: Command=GetFoldersAndFiles&Type=File&CurrentFolder=/Samples/Docs/
 *  response:
 *  <?xml version="1.0" encoding="utf-8" ?>
 *  <Connector command="GetFoldersAndFiles" resourceType="File">
 *   <CurrentFolder path="/Samples/Docs/" url="/UserFiles/File/Samples/Docs/" />
 *   <Folders>
 *     <Folder name="Documents" />
 *     <Folder name="Files" />
 *     <Folder name="Other Files" />
 *     <Folder name="Related" />
 *   </Folders>
 *   <Files>
 *     <File name="XML Definition.doc" size="14" />
 *     <File name="Samples.txt" size="5" />
 *     <File name="Definition.txt" size="125" />
 *     <File name="External Resources.drw" size="840" />
 *     <File name="Todo.txt" size="2" />
 *   </Files>
 *  </Connector>
 *
 *  request:Command=CreateFolder&Type=File&CurrentFolder=/Samples/Docs/&NewFolderName=FolderName
 *  response:
```

```
 *  <?xml version="1.0" encoding="utf-8" ?>
 *  <Connector command="CreateFolder" resourceType="File">
 *    <CurrentFolder path="/Samples/Docs/" url="/UserFiles/File/Samples/Docs/" />
 *    <Error number="0" />
 *  </Connector>
 *
 *  request: Command=FileUpload&Type=File&CurrentFolder=/Samples/Docs/
 *  response: (Simple HTML):
 *  <script type="text/javascript">
 *       window.parent.frames['frmUpload'].OnUploadCompleted(0) ;
 *  </script>
 *  #OnUploadCompleted( 0 ) : no errors found on the upload process.
 *  #OnUploadCompleted( 201, 'FileName(1).ext' ) : the file has been uploaded successfully, but its name has
been changed to "FileName(1).ext".
 *  #OnUploadCompleted( 202 ) : invalid file.
 *
 *    Possible Error Numbers are:
 *       0 : No Errors Found. The folder has been created.
 *     101 : Folder already exists.
 *     102 : Invalid folder name.
 *     103 : You have no permissions to create the folder.
 *     110 : Unknown error creating folder.
 */

public class FckAction extends AbstractAction {
    private final static String LOGHEADER = "com.zamek.portal_zamek.modules.actions.";

    protected final static String CTX_COMMAND = "Command";
    protected final static String CTX_TYPE = "Type";
    protected final static String CTX_CURRENT_FOLDER = "CurrentFolder";
    protected final static String CTX_NEW_FOLDER = "NewFolderName";
    protected final static String CTX_NEW_FILE = "NewFile";
    protected final static String FT_FILE = "File";
    protected final static String FT_IMAGE = "Image";
    protected final static String FT_FLASH = "Flash";
    protected final static String FT_MEDIA = "Media";
    protected final static String CTX_FILE_SERV_ACTION = "FileServ";

    private final static String CMD_GET_FOLDERS = "GetFolders";
    private final static String CMD_GET_FOLDERS_AND_FILES = "GetFoldersAndFiles";
    private final static String CMD_CREATE_FOLDER = "CreateFolder";
    private final static String CMD_FILE_UPLOAD = "FileUpload";
    protected final static String XML_CONNECTOR = "Connector";
    protected final static String XML_COMMAND = "command";
    protected final static String XML_CURRENT_FOLDER = "CurrentFolder";
    protected final static String XML_PATH = "path";
    protected final static String XML_URL = "url";
    protected final static String XML_RESOURCE_TYPE = "resourceType";
    protected final static String XML_FOLDERS = "Folders";
    protected final static String XML_FOLDER = "Folder";
    protected final static String XML_FILES = "Files";
    protected final static String XML_FILE = "File";
    protected final static String XML_NAME = "name";
    protected final static String XML_SIZE = "size";
    protected final static String XML_ERROR = "Error";
    protected final static String XML_NUMBER = "Number";

    protected final static int KILO_BYTE= 1024;
    protected final static String CTX_RESPONSE="response";

    protected final static int ERR_FOLDER_EXISTS =     101; // : Folder already exists.
    protected final static int ERR_INVALID_FOLDER = 102; // : Invalid folder name.
    protected final static int ERR_NO_PERMISSION = 103;  // : You have no permissions to create the folder.
    protected final static int ERR_UNKNOWN = 110;          // : Unknown error creating folder.

    protected final static int RES_UPLOAD_INVALID = 202;
    protected final static int RES_UPLOAD_OK = 0;
    protected final static int RES_UPLOAD_NAME_CHANGED = 201;

    protected final static String UPLOAD_OK_RESULT =
        "<script type=\"text/javascript\">window.parent.frames['frmUpload'].OnUploadCompleted(0,\"%s\");<
```

```
/script>";
    protected final static String UPLOAD_ERROR_RESULT =
        "<script type=\"text/javascript\">window.parent.frames['frmUpload'].OnUploadCompleted(%d);</script>";

    protected String command = null,
                     type = null,
                     currentFolder = null,
                     scriptResult = null,
                     repo = null,
                     id = null;

    protected RunData data;

    protected Document document = null;
    protected File serverDir = null;
    protected Node rootNode = null;

    public void doPerform(RunData data, Context ctx) throws Exception {
        this.data = data;
        data.declareDirectResponse();
        data.setLayout(ZamekConst.DIRECT_RESPONSE_LAYOUT);
        processRequest(data);
    }

    private void processRequest(RunData data) throws ServletException, IOException {
        HttpServletResponse response = data.getResponse();
        try {
            ParameterParser pp = data.getParameters();
            command = pp.get(CTX_COMMAND);
            type =  pp.get(CTX_TYPE);
            currentFolder = pp.get(CTX_CURRENT_FOLDER);
            repo = pp.get(DirectoryService.CTX_REPO);
            id = pp.get(DirectoryService.CTX_ID);

            if (command == null || command.length() == 0 ||   // possible abuse
                repo == null || repo.length() == 0 ||
                id == null || id.length() == 0)
                return;

            initDocument();

            rootNode = createCommonXml();
            doCommand();
            if (rootNode == null) {  // result of upload is a javascipt code
                response.setContentType("text/html");
                response.getOutputStream().print(scriptResult);
                return;
            }
            response.setContentType("text/xml");
            document.getDocumentElement().normalize();
            try {
                System.setProperty ( "javax.xml.transform.TransformerFactory" ,
                        "org.apache.xalan.xsltc.trax.TransformerFactoryImpl");
                TransformerFactory tFactory = TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer();
                DOMSource source = new DOMSource(document);
                StreamResult result = new StreamResult(new ByteArrayOutputStream ());
                transformer.transform(source, result);
                String xml = result.getOutputStream().toString();
                response.getOutputStream().print(xml);
            }
            catch (Exception e) {
                errorReport (data, LOGHEADER, e);
            }

        }
        catch (Exception e) {
            log.error(LOGHEADER+" error:"+e.getMessage());
        }
    }
```

```
protected void initDocument () throws ParserConfigurationException {
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    document = builder.newDocument();
}

protected Node createCommonXml () {
       Element root = document.createElement(XML_CONNECTOR);
       document.appendChild(root);
       root.setAttribute(XML_COMMAND, command);
       root.setAttribute(XML_RESOURCE_TYPE, type);

       Element el = document.createElement(XML_CURRENT_FOLDER);
       el.setAttribute(XML_PATH, currentFolder);

    TemplateURI tu = new TemplateURI();
    tu.setScreen(CTX_FILE_SERV_ACTION);
    tu.addPathInfo(DirectoryService.CTX_REPO, repo);
    tu.addPathInfo(DirectoryService.CTX_ID, id);
    String fn = currentFolder.replace('/', ',');
    tu.addPathInfo(DirectoryService.CTX_FILE, fn);
    el.setAttribute(XML_URL, tu.toString());
       root.appendChild(el);
       return root;
}

protected void doCommand() throws Exception {
    if (command.compareTo(CMD_GET_FOLDERS_AND_FILES) == 0) {
        getFoldersAndFiles (rootNode);
        return;
    }
    if (command.compareTo(CMD_GET_FOLDERS) == 0) {
        getFolders(rootNode);
        return;
    }
    if (command.compareTo(CMD_CREATE_FOLDER) == 0) {
        createFolder (rootNode);
        return;
    }
    if (command.compareTo(CMD_FILE_UPLOAD) == 0) {
        scriptResult = fileUpload();
        rootNode = null;
    }
}

protected String getFileTypeMask() {
    if (type.compareTo(FT_FILE) == 0)
        return null;
    if (type.compareTo(FT_IMAGE) == 0)
        return ".*\\.gif|.*\\.jpg|.*\\.jpeg|.*\\.png";
    if (type.compareTo(FT_FLASH) == 0)
        return ".*\\.swf";
    if (type.compareTo(FT_MEDIA) == 0)
        return ".*\\.vmw|.*\\.mov";
    return null;
}

protected void getFoldersAndFiles(Node root) throws Exception {
    getFolders(root);
    List<FileInfo> result = getFoldersAndFileList();
    Element files = document.createElement(XML_FILES);
    root.appendChild(files);
    if (result != null && result.size() > 0) {
        for (Iterator<FileInfo> it=result.iterator(); it.hasNext(); ) {
            Element file = document.createElement(XML_FILE);
            FileInfo fi = it.next();
            files.appendChild(fi.setElement(file, KILO_BYTE));
        }
    }
}
```

```
protected void getFolders(Node root) throws Exception {
    List<String> result = getFoldersList();
     Element folders = document.createElement(XML_FOLDERS);
    root.appendChild(folders);
    if (result != null && result.size() > 0) {
       for (Iterator<String> it=result.iterator(); it.hasNext(); ) {
           Element subDir = document.createElement(XML_FOLDER);
           subDir.setAttribute(XML_NAME, it.next());
           folders.appendChild(subDir);
       }
    }
}

protected List<FileInfo> getFoldersAndFileList() throws Exception
{
    return TurbineDirectory.getFiles(repo, id, currentFolder, getIncludeMask(), getExcludeMask());
}

protected List<String> getFoldersList() throws Exception
{
    return TurbineDirectory.getFolders(repo, id, currentFolder, getIncludeMask(), getExcludeMask());
}

protected String getBaseUrl(RunData data) throws Exception
{
    return repo+'/'+id;
}

protected int createFolder(Node root) throws Exception
{
    String newFolder = data.getParameters().get(CTX_NEW_FOLDER);
    int result = canCreateFolder() && newFolder != null
        ? TurbineDirectory.createDirectory(repo, id, newFolder)
        : DirectoryService.ERR_NO_PERMISSION;

    Element crf = document.createElement(XML_ERROR);
    crf.setAttribute(XML_ERROR, Integer.toString(result));
    root.appendChild(crf);
    return result;
}

protected String fileUpload() throws Exception
{
    int result = RES_UPLOAD_INVALID;
    if (canUpload())
    {
        FileItem fItem = data.getParameters().getFileItem(CTX_NEW_FILE);
        if (fItem != null)
        {
            try
            {
                if (TurbineDirectory.createIdDirectory(repo, id) &&
                    TurbineDirectory.uploadItemFile(repo, id, currentFolder, fItem))
                    return String.format(UPLOAD_OK_RESULT, fItem.getName());
            }
            catch (Exception e) {
                log.error(LOGHEADER+"fileUpload error"+e.getMessage());
            }
        }

    }
    return String.format(UPLOAD_ERROR_RESULT, result);
}

/* overridable methods */

/**
 * return rights to can create directory
 * RunData is readable by protected data variable
 */
protected boolean canCreateFolder()
```

```
    {
        return data.getUser().hasLoggedIn() && TurbineDirectory.getCreateDirEnabled(repo);
    }

    /**
     * return rights to can upload
     * RunData is readable by protected data variable
     */
    protected boolean canUpload () {
        return data.getUser().hasLoggedIn();
    }

    /**
     * getFolders() an getFolderAndFileList() using to set including name mask
     * null is similar to *
     * default value is null
     * @return
     */
    protected String getIncludeMask()
    {
        return null;
    }

    /**
     * getFolders() an getFolderAndFileList() using to set excluding name mask
     * null is similar to nothing
     * default is null
     * @return
     */
    protected String getExcludeMask()
    {
        return null;
    }

}
```

serving files to requests

We need to make a screen module to serve file requests. It is a descendants of_* 'R'***aw***'S'***creen. We cannot use / as path separator in url because Turbine has a special url encoding method named path_info which is name/value format. We have to change / to , in url path parameters. Code of*** 'F'***ile***'S'*_erv.java is:

```
package com.zamek.portal_zamek.modules.screens;

import java.io.ByteArrayOutputStream;

import javax.activation.MimetypesFileTypeMap;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.turbine.modules.screens.RawScreen;
import org.apache.turbine.util.RunData;

import com.zamek.portal_zamek.services.directory.DirectoryService;
import com.zamek.portal_zamek.services.directory.TurbineDirectory;

public class FileServ extends RawScreen {
    private final static String LOGHEADER = "com.zamek.portal_zamek.modules.screens.FileServ.";
    protected Log log = LogFactory.getLog(this.getClass());

    @Override
    protected void doOutput(RunData data) throws Exception {
        if (log.isDebugEnabled())
            log.debug(LOGHEADER + "doOutput, enter");
        try {
            String repo = data.getParameters().get(DirectoryService.CTX_REPO),
                   id = data.getParameters().get(DirectoryService.CTX_ID),
                   fileName = data.getParameters().get(DirectoryService.CTX_FILE);
            if (repo == null || id == null || fileName == null)
                return;


            String fName = fileName.replace(',', '/'); // restore / in url

            // private files only for hasloggedin users
            if (fName.startsWith(DirectoryService.MASK_PRIVATE) &&
                 ! data.getUser().hasLoggedIn() )
                return;

            ByteArrayOutputStream bos = TurbineDirectory.getFile(repo, id, fName);
            if (bos == null)
                return;
            bos.writeTo(data.getResponse().getOutputStream());
        }
        catch (Exception e) {
            log.error(LOGHEADER + "doOutput error:" + e.getMessage());
        }
    }

    @Override
    protected String getContentType(RunData data) {
        String fileName = data.getParameters().get(DirectoryService.CTX_FILE);
        if (StringUtils.isEmpty(fileName))
            return null;
        String ext = fileName.substring(fileName.lastIndexOf('.'));
        return new MimetypesFileTypeMap().getContentType(ext);
    }

}
```

setting FCKEditor in Velocity forms

The best way is to make two macros for setting FCKEditor:

```
#macro (declareFCK)
    $page.addScript($content.getURI("scripts/FCKeditor/fckeditor.js"))
#end
```

which is initialize editor, and

```
#macro (textAreaFckField $name $value $height $toolbarSet $action $repo $id)
    ##pre("name:$name value:$value height:$height toolbarset:$toolbarSet action:$action, repo:$repo, id:$id")
    <textarea id="$name" class="frminput" name="$name">$!value</textarea>
    <script type="text/javascript">
        #if ($height)
              var oFCKeditor = new FCKeditor( '$name','100%','$height', 'Default', '') ;
        #else
              var oFCKeditor = new FCKeditor( '$name' ) ;
        #end
          oFCKeditor.BasePath = "$content.getURI('scripts/FCKeditor/')";
        #if ($toolbarSet)
            oFCKeditor.ToolbarSet = "$toolbarSet";
        #else
            oFCKeditor.ToolbarSet = "Default" ;
        #end
        #if ($action.length() != 0)
            #set($connector=$link.setAction($action).addPathInfo("repo", $repo).addPathInfo("id",$id).
toString())
            oFCKeditor.Config.LinkBrowserURL = oFCKeditor.BasePath + "editor/filemanager/browser/default
/browser.html?Connector=$connector";
            oFCKeditor.Config.ImageBrowserURL = oFCKeditor.BasePath + "editor/filemanager/browser/default
/browser.html?Connector=$connector/Type/Image";
            oFCKeditor.Config.FlashBrowserURL = oFCKeditor.BasePath + "editor/filemanager/browser/default
/browser.html?Connector=$connector/Type/Flash";
        #else
            oFCKeditor.Config.LinkBrowserURL = null;
            oFCKeditor.Config.ImageBrowserURL = null;
            oFCKeditor.Config.FlashBrowserURL = null;
        #end
        oFCKeditor.Config.AutoDetectLanguage = false ;
        oFCKeditor.Config.DefaultLanguage = "hu" ;
        oFCKeditor.ReplaceTextarea ();
    </script>
#end
```

which is a form field for FCKEditor parameters are:

- name: name of field
- value: value of field
- height: height of_* 'FC'***KE***'ditor in pixels
- toolbarset: there are different toolbarset, "" will be "Default" set
- action: name of action to handle '***FC***'KE'***ditor requests, default is*** 'F'***ck***'A'*_ction
- repo: name of sub repository, for example user. It is mapping under resources/ directory (services.DirectoryService.repository in TR.props)
- id: id of identity in repo. for example user's id. It will mapping under resources/user/42 if user's id is 42

Let's see a form example:

```
#headerTitle("User's data")
$data.setLayoutTemplate("Default.vm")
#declareFCK()
<table width="100%" border="0">
...
<form name="userreg" method="post" action='$link.setAction("UserReg").toString()'>
...
        #textAreaFckField("intro" $showUser.Intro 500 "Default" "UserFile" "user" $showUser.UserId)
...
</form>
...
</table>
```

So, it works, but you need some utilities to complete First is_* 'F'***ile***'I'*_nfo class:

```
package com.zamek.portal_zamek.util;

import org.w3c.dom.Element;

/**
 * simple class to serve file informations for TurbineirectoryService
 * @author zamek
 *
 */
public class FileInfo {

        private String name;
        private long size;

        public final static String XML_NAME = "name";
        public final static String XML_SIZE = "size";

        public FileInfo (String name, long size) {
                this.name = name;
                this.size = size;
        }

        public String getName () {
                return name;
        }

        public long getSize () {
                return size;
        }

        public Element setElement (Element element, long divider) {
                if (element == null)
                        return null;
                element.setAttribute(XML_NAME, name);
                element.setAttribute(XML_SIZE,
                                Long.toString(
                                                divider > 1 ? size / divider : size));
                return element;
        }
}
```

and the last (I promice!) class is_* 'F'***ile***'U_'_'til:

```
package com.zamek.portal_zamek.util;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.List;
import java.util.Vector;
import org.apache.commons.fileupload.FileItem;

import com.zamek.portal_zamek.PortalException;
import com.zamek.portal_zamek.ZamekConst;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

/**
 * @author zamek Created 2004.05.07.16:18:41
 * @version $Id: FileUtils.java,v 1.2 2006/02/07 15:25:12 zamek Exp $
 */
public class FileUtils implements ZamekConst {
        private final static String LOGHEADER = "com.zamek.portal_zamek.util.FileUtils.";
```

```
protected static Log log = LogFactory.getLog(FileUtils.class);

public final static int CMD_OK_EXITVALUE = 0;

public final static String BIN_DIRECTORY = "/bin/";

protected final static int MAX_BUFFER_SIZE = 1024;

/**
 * Execute a system shell command
 *
 * @param path
 *            path of command
 * @param command
 *            command
 * @param params
 *            parameters of command
 * @throws PortalException
 * @throws IOException
 * @throws InterruptedException
 */
public static int executeShell(final String path, final String command, final String[] params)
                throws PortalException, IOException, InterruptedException {
if (command == null || command.length() == 0)
    throw new PortalException (MSG_EMPTY_COMMAND);
        StringBuffer cmd = new StringBuffer(pathEnd(path));
        cmd.append(command);
        if (params != null && params.length > 0)
                for (int i = 0; i < params.length; i++)
                        cmd.append(" ").append(params[i]);
        Process process = Runtime.getRuntime().exec(cmd.toString());
        process.waitFor();
        if (process.exitValue() != CMD_OK_EXITVALUE)
                throw new PortalException(MSG_ERROR_EXIT + process.exitValue());
        return process.exitValue();
}

/**
 * Make a directory from path. Path must be exists!
 *
 * @param path
 *            parent path
 * @param newDir
 *            name of new directory
 * @throws PortalException
 */
public static String makeDir(String path, final String newDir)
                throws PortalException {
        File file = new File(path);

if (! file.isDirectory() || ! file.exists() )
    throw new PortalException(MSG_PARENT_NOTEXISTS + path);

        path = pathEnd(path);

        String dirName = path + newDir;
        file = new File(dirName);
        if (file.exists())
                return dirName + "/";
        file.mkdir();
        return dirName + "/";
}

/**
 * Make a named directory
 * @param newDir path and name of new directory
 */
public static void makeDir (final String newDir) {
        File file = new File (newDir);
        if (file.exists())
                return;
```

```
                file.mkdir();
        }

        /**
         * getting size of file or directory
         * @param file File object representing a file or Directory
         * @return size of file or Directory
         */
        public static long getSize (File file) {
                if (file == null)
                        return 0;
                if (file.isFile())
                        return file.length();
                File[] files = file.listFiles();
                if (files == null)
                        return 0;
                long size = 0;
                for (int i=0; i<files.length; i++)
                        size += getSize (files[i]);
                return size;
        }

        /**
         * getting size of file or Directory
         * @param name String name of file or directory
         * @return size of file or directory
         */
        public static long getSize(final String name) {
                File file = new File(name);
                return getSize (file);
        }


        /**
    * Remove directory
         *
         * @param dir
         *             name of deleting directory
         * @param force
         *            force delete if not empty
         * @throws PortalException
         * @throws IOException
         */
        public static void removeDir(final String dir, boolean force)
                        throws PortalException, IOException {
                File file = new File(dir);
        if (!file.isDirectory())
            throw new PortalException (MSG_DIR_NOT_EXISTS);
                File[] files = file.listFiles();
        if (!force && files.length > 0)
            throw new PortalException(MSG_DIR_NOT_EMPTY);
                if (force && files.length > 0) {
                        for (int i = 0; i < files.length; i++)
                                files[i].delete();
                }
                file.delete();
        }

        /**
         * delete directory if it is empty
         *
         * @param dir
         *            name of deleteing directory
         * @throws PortalException
         * @throws IOException
         */
        public void removeDir(final String dir) throws PortalException, IOException {
                removeDir(dir, false);
        }

        /**
```

```
     * append pathSeparator after path if it is not exists
     *
     * @param path
     * @return
     */
    public static String pathEnd(final String path) {
            return path.charAt(path.length() - 1) == File.separatorChar ? path
                            : path + File.separatorChar;
    }

    /**
     * return less long value
     *
     * @param a
     * @param b
     * @return
     */
    public static long min(long a, long b) {
            return Math.min(a, b);
    }

    /**
     * return greates long value
     *
     * @param a
     * @param b
     * @return
     */
    public static long max(long a, long b) {
            return Math.max(a, b);
    }

    /**
     * return less int value
     *
     * @param a
     * @param b
     * @return
     */
    public static int min(int a, int b) {
            return Math.min(a, b);
    }

    /**
     * return greatest int value
     *
     * @param a
     * @param b
     * @return
     */
    public static int max(int a, int b) {
            return Math.max(a, b);
    }

    /**
     * delete file
     *
     * @param name
     * @throws PortalException
     * @throws IOException
     */
    public static void deleteFile(final String name) throws PortalException,
                    IOException {
            File file = new File(name);
    if (!file.exists())
        throw new PortalException(MSG_FILE_NOT_EXISTS + name);
            file.delete();
    }

/**
 * secure file delete. It must be setting separate path and filename.
```

```
 * remove beginning separatorChar from name. (it cannot be delete from /)
 * @param path
 * @param file
 * @throws PortalException
 * @throws IOException
 */
public static void deleteFile(String path, String file) throws PortalException, IOException {
    path = pathEnd(path);
    while (file.charAt(0) == File.separatorChar)
        file = file.substring(1);
    File f = new File (new StringBuffer(path).append(file).toString());
    if (!f.exists())
        throw new PortalException(new StringBuffer (MSG_FILE_NOT_EXISTS).append(f.getName()).toString());
    f.delete();
}

    /**
     * copy file.
 * file másolása. Ha targetCheck true és létezik a cél file,
     * PortalException-t generál
     *
     * @param source
     *            source (include path)
     * @param dest
     *            target (include path)
     * @param targetCheck
     *            checking target before copy.
 *            Throws an PortalException if it is true and target is exists
     * @throws PortalException
     * @throws IOException
     */
    public static void copyFile(final String source, final String dest, boolean targetCheck)
                    throws PortalException, IOException {
            if (log.isDebugEnabled())
                    log.debug(LOGHEADER + "copyFile source:"+source+", dest:"+dest);

    File src = new File(source);
    if (!src.exists())
        throw new PortalException(MSG_FILE_NOT_EXISTS + source);

    File dst = new File(dest);
            if (targetCheck && dst.exists())
        throw new PortalException(MSG_TARGET_EXISTS + dest);

    FileInputStream fis = new FileInputStream(src);
            FileOutputStream fos = new FileOutputStream(dst);
            byte[] buffer = new byte[Math.min((int) src.length(), MAX_BUFFER_SIZE)];
            while (fis.read(buffer) != -1)
                    fos.write(buffer);
            fis.close();
            fos.close();
    }

    /**
     * copy file. It doesn't check exists of target
     *
     * @param source
     *            source (include path)
     * @param dest
     *            target (include path)
     * @throws PortalException
     * @throws IOException
     */
    public static void copyFile(final String source, final String dest)
                    throws PortalException, IOException {
            copyFile(source, dest, false);
    }

    /**
     * move file.
 * file mozgatása. Ha targetCheck true és létezik a cél file,
```

```
     * PortalException-t generál
     *
     * @param source
     *            source (include path)
     * @param dest
     *            target (include path)
     * @param targetCheck
*            checking target before copy.
*            Throws an PortalException if it is true and target is exists
     * @throws PortalException
     * @throws IOException
     */
    public static void moveFile(final String source, final String dest, boolean targetCheck)
                    throws PortalException, IOException {
            copyFile(source, dest, targetCheck);
            deleteFile(source);
    }

    /**
     * move file. it doesn't check exists of target
     *
     * @param source
     *            source (include path)
     * @param dest
     *            target (include path)
     * @throws PortalException
     * @throws IOException
     */
    public static void moveFile(String source, String dest)
                    throws PortalException, IOException {
            moveFile(source, dest, false);
    }

    /**
     * copy uploaded file to toDir with newName.
     * @param fileItem
     *            upload fileItem
     * @param toDir
     *            target directory
     * @param newName
     *            name of new fileName
     * @throws PortalException
     *             if fileItem == null || fielItem.size() == 0
     * @throws Exception
     */
    public static String uploadFile(FileItem fileItem, final String toDir,
                    final String newName) throws PortalException, Exception {
            if (log.isDebugEnabled())
                            log.debug(LOGHEADER + "uploadFile enter, todir:"+toDir+
                      ",newname:"+newName);
    if (fileItem == null || fileItem.getSize() == 0)
        throw new PortalException(MSG_FILE_NULL_OR_EMPTY);
    StringBuffer sb = new StringBuffer(pathEnd(toDir)).append(newName);
            fileItem.write(new File(sb.toString()));
            return sb.toString();
    }

    /**
     * separate filename into path, name, ext
     *
     * @param source
     *            input filaname
     * @return [0] : path, [1]: name, [2]: ext
     */
    public static String[] stripFileName(final String source) {
            String[] res = new String[3];

            int extPos = source.lastIndexOf('.');
            res[2] = extPos > 0 ? source.substring(extPos) : EMPTY_STR;

            String src = source.substring(0, extPos);
```

```
    int namePos = source.lastIndexOf(File.separatorChar);
            if (namePos >= 0) {
                    res[1] = src.substring(++namePos);
                    res[0] = src.substring(0, namePos);
            }
    else {
                    res[0] = EMPTY_STR;
                    res[1] = EMPTY_STR;
            }
            return res;
    }

/**
 * get extent of filename
 * @param name
 * @return
 */
    public static String getFileExt(final String name) {
            int extPos = name.lastIndexOf('.');
            return extPos > 0 ? name.substring(extPos) : EMPTY_STR;
    }

    /**
     * extract name from absolutePath+fileName
     * @param fileName
     * @return
     */
    public static String getFileName(final String fileName) {
            if (StringUtils.isEmpty(fileName))
                    return fileName;
            int namePos = fileName.lastIndexOf(File.pathSeparator);
            return namePos == -1
                    ? fileName
                    : fileName.substring(namePos);
    }
    /**
     * list content of directory. It can filter with optional parameters
     * @param dir
     *            listing directory. It check exists, readable, directory check
     * @param mask
 *            Optional filter mask. It masks with String.matches()
     * @return list of files
     */
    public static List<String> getFilesInDirectory(final String dir, final String mask) {
            String[] files;
            Vector<String> result = new Vector<String>();
            File directory = new File(dir);
            if (directory.exists() && directory.canRead()
                            && directory.isDirectory()) {
                    files = directory.list();
                    if (mask != null && mask.length() > 0) {
                            for (int i = 0; i < files.length; i++) {
                                    String fName = files[i];
                                    if (fName.matches(mask))
                                            result.add(fName);
                            }
                    }
        else {
                            for (int i = 0; i < files.length; i++)
                                    result.add(files[i]);
                    }
                    return result;
            }
            return null;
    }

    public static List<String> getDirectoryNames (final String dir,
                                    final String includeMask, final String excludeMask) {
            String[] files;
            Vector<String> result = new Vector<String>();
```

```
                File directory = new File(dir);
                if (directory.exists() && directory.canRead()
                                && directory.isDirectory()) {
                        files = directory.list();
                        for (int i=0; i < files.length; i++) {
                                String cName = files[i];
                                File subDir = new File (dir + File.separatorChar + cName);
                                if (! subDir.isDirectory())
                                        continue;
                                if (includeMask == null && excludeMask == null) {
                                        result.add(cName);
                                        continue;
                                }
                                if (excludeMask != null && cName.matches(excludeMask))
                                        continue;
                                if (includeMask != null && cName.matches(includeMask))
                                        result.add(cName);
                        }
                        return result;
                }
                return null;
        }

        public static List<FileInfo> getFileNames (final String dir,
                                        final String includeMask, final String excludeMask) {
                String[] files;
                Vector<FileInfo> result = new Vector<FileInfo>();
                File directory = new File(dir);
                if (directory.exists() && directory.canRead()
                                && directory.isDirectory()) {
                        files = directory.list();
                        for (int i=0; i < files.length; i++) {
                                String cName = files[i];
                                File file = new File (dir + File.separatorChar + cName);
                                if (! file.isFile())
                                        continue;
                                if (includeMask == null && excludeMask == null) {
                                        result.add(new FileInfo(cName, file.length()));
                                        continue;
                                }
                                if (excludeMask != null && cName.matches(excludeMask))
                                        continue;
                                if ( (includeMask == null) || (includeMask != null && cName.matches
(includeMask)) ) {
                                        result.add(new FileInfo(cName, file.length()));

                                }
                        }
                        return result;
                }
                return null;

        }

        /**
         * list content of directory. It can filter with optional parameters
         * @param dir
         *            listing directory. It check exists, readable, directory check
         * @param includeMask
     *            Optional filter mask. It masks with String.matches()
         * @param excludeMask
     *            Optional filter mask. It masks with String.matches()
         * @return list of files
         */
        public static List getFilesInDirectory(final String dir, final String includeMask, final String
excludeMask) {
                String[] files;
                Vector<String> result = new Vector<String>();
                File directory = new File(dir);
                boolean chkInclude = includeMask != null && includeMask.length() > 0,
                                chkExclude = excludeMask != null && excludeMask.length() > 0;
```

```
            if (directory.exists() && directory.canRead()
                            && directory.isDirectory()) {
                    files = directory.list();
                    for (int i = 0; i < files.length; i++) {
                            String fName = files[i];
                            if ( (chkExclude && (! fName.matches(excludeMask)) &&
                                      chkInclude && fName.matches(includeMask)))
                                    result.add(fName);
                            else
                                    if (chkInclude && fName.matches(includeMask))
                                            result.add(fName);
                                    else
                                            result.add(fName);
                    }
                    return result;
            }
            return null;
        }


    /**
     * checks end of name with needExtent
     * @param name
     * @param needExtent
     * @return
     */
    public static boolean checkExtent (final String name, final String needExtent) {
       return needExtent != null && name != null &&
              name.length() >= needExtent.length() &&
              name.endsWith(needExtent);
    }

        final static char BACK_SLASH = '\\',
                                       SLASH = '/',
                                       COLON = ':',
                                       SPACE = ' ',
                                       UNDER_LINE = '_';
        final static char [] INTERNATIONALS = {'á','í','','','ü','ö','ú','ó','é',

'Á','Í','','','Ü','Ö','Ú','Ó','É',
                                                                                '?','?','?','?'};
        final static char[] ASCII = {'a','i','u','o','u','o','u','o','e',
                                                                'A','I','U','O','U','O','U','O','E',
                                                                'u','o','U','O'};


        /**
         * Normalize file name.
         * replace ':' to '_', '\' to '/',
         * international chars to ascii representant or '_'
         *
         * @param s
         * @return normalized filename
         */
        public static String normalizeFileName(String s) {
                if (StringUtils.isEmpty(s))
                        return s;
                if (s.indexOf(BACK_SLASH) > 0 && s.indexOf(COLON) > 0)
                        s = processWinFileName (s);
                s = s.replace(SPACE, UNDER_LINE);
                return changeInternationalChars(s);
        }

        private static String processWinFileName(String s) {
                s= s.replace(COLON, UNDER_LINE);
                return s.replace(BACK_SLASH, SLASH);
        }

        private static String changeInternationalChars(String s) {
                for (short i=0; i<s.length(); i++) {
                        char at = s.charAt(i);
```

```
                        if (at > 127) {
                                for (short j=0; j<INTERNATIONALS.length; j++) {
                                        if (at == INTERNATIONALS[j]) {
                                                s = s.replace(at, ASCII [j]);
                                                break;
                                        }
                                }
                                s = s.replace(at, UNDER_LINE);
                        }
                }
                return s;
        }

}
```