EVIDEN

Identity and Access Management

DirX Identity

Web Center File Upload

Version 8.10.15, Edition April 2026

Table of Contents

Preface

This document presents a use case that describes DirX Identity Web Center's file upload feature. It consists of the following chapters:

- Chapter 1 provides an overview of Web Center's file upload capabilities and the sample use cases.
- Chapter 2 describes the Web Center file upload user interface.
- Chapter 3 describes how the Web Center file upload feature is configured.

DirX Identity Documentation Set

*Version 8.10.15 | Build 1932 | Date 2026-04-30 *

The DirX Identity document set consists of the following manuals:

- *DirX Identity Introduction*. Use this book to obtain a description of DirX Identity architecture and components.
- *DirX Identity Release Notes*. Use this book to understand the features and limitations of the current release. This document is shipped with the DirX Identity installation as the file **release-notes.pdf**.
- *DirX Identity History of Changes*. Use this book to understand the features of previous releases. This document is shipped with the DirX Identity installation as the file **history-of-changes.pdf**.
- *DirX Identity Tutorial*. Use this book to get familiar quickly with your DirX Identity installation.
- *DirX Identity Provisioning Administration Guide*. Use this book to obtain a description of DirX Identity provisioning architecture and components and to understand the basic tasks of DirX Identity provisioning administration using DirX Identity Manager.
- *DirX Identity Connectivity Administration Guide*. Use this book to obtain a description of DirX Identity connectivity architecture and components and to understand the basic tasks of DirX Identity connectivity administration using DirX Identity Manager.
- *DirX Identity User Interfaces Guide*. Use this book to obtain a description of the user interfaces provided with DirX Identity.
- *DirX Identity Application Development Guide*. Use this book to obtain information how to extend DirX Identity and to use the default applications.
- *DirX Identity Customization Guide*. Use this book to customize your DirX Identity environment.
- *DirX Identity Integration Framework*. Use this book to understand the DirX Identity framework and to obtain a description how to extend DirX Identity.
- *DirX Identity Web Center Reference*. Use this book to obtain reference information about the DirX Identity Web Center.
- *DirX Identity Web Center Customization Guide*. Use this book to obtain information how to customize the DirX Identity Web Center.
- *DirX Identity Meta Controller Reference*. Use this book to obtain reference information about the DirX Identity meta controller and its associated command-line programs and files.
- *DirX Identity Connectivity Reference*. Use this book to obtain reference information about the DirX Identity agent programs, scripts, and files.
- *DirX Identity Troubleshooting Guide*. Use this book to track down and solve problems in your DirX Identity installation.
- *DirX Identity Installation Guide*. Use this book to install DirX Identity.

- *DirX Identity Migration Guide*. Use this book to migrate from previous versions.

Notation Conventions

Boldface type
In command syntax, bold words and characters represent commands or keywords that must be entered exactly as shown.

In examples, bold words and characters represent user input.

Italic type
In command syntax, italic words and characters represent placeholders for information that you must supply.

[ ]
In command syntax, square braces enclose optional items.

{ }
In command syntax, braces enclose a list from which you must choose one item.

In Tcl syntax, you must actually type in the braces, which will appear in boldface type.

|
In command syntax, the vertical bar separates items in a list of choices.

...
In command syntax, ellipses indicate that the previous item can be repeated.

userID_home_directory
The exact name of the home directory. The default home directory is the home directory of the specified UNIX user, who is logged in on UNIX systems. In this manual, the home pathname is represented by the notation *userID_home_directory*.

install_path
The exact name of the root of the directory where DirX Identity programs and files are installed. The default installation directory is *userID_home_directory*/**DirX Identity** on UNIX systems and **C:\Program Files\DirX\Identity** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathnames is represented by the notation *install_path*.

dirx_install_path
The exact name of the root of the directory where DirX programs and files are installed. The default installation directory is *userID_home_directory*/**DirX** on UNIX systems and **C:\Program Files\DirX** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathname is represented by the notation *dirx_install_path*.

dxi_java_home
The exact name of the root directory of the Java environment for DirX Identity. This location is specified while installing the product. For details see the sections "Installation" and "The Java for DirX Identity".

tmp_path

The exact name of the tmp directory. The default tmp directory is /tmp on UNIX systems. In this manual, the tmp pathname is represented by the notation *tmp_path*.

tomcat_install_path
The exact name of the root of the directory where Apache Tomcat programs and files are installed. This location is defined during product installation.

mount_point
The mount point for DVD device (for example, **/cdrom/cdrom0**).

1. Overview

DirX Identity Web Center's file upload feature supports the following actions:

- Uploading one or more files to the Web server.
- Storing the files in a folder on the server.
- Starting a handler program to process uploaded files.
- Displaying status records produced by handler programs.

An upload page might allow for uploading just a single file, or for uploading multiple files. You can define the maximum number of files to upload in a single step. You can also set a limit for the overall file size to prevent users from uploading huge files. The files must also pass various checks to prevent malicious content to be uploaded to the server.

File upload is governed by menu access policies. These policies control whether a user gets the menu items to upload files. They are also checked when a user attempts to upload files to prevent unauthorized users from uploading files by bypassing the Web Center menu.

File upload can be temporarily blocked on the server side; for example, to allow an operator to perform some cleanup or management tasks on the upload folder.

The uploaded files are stored in a folder on the server. You can define the name of the folder, the names and extensions of the target files, and whether existing files are overwritten. To avoid naming conflicts and to be able to easily map files to users, the target file name template can include placeholders for the uploading user's DN, one or more of his attributes (like last name and first name), a timestamp, the original file name, and a counter.

After a file has been uploaded and stored on the server, Web Center can start a program to process the file. The program must be provided by the customer. It is started per uploaded file. You can configure the program name, the list of arguments to be passed to the program, its working directory and its standard input, output, and error file. Web Center just starts the program; it doesn't wait for it to finish. The handler programs also run independently of the Tomcat process; for example, they are not affected by Tomcat restarts.

A handler program can write status records about its progress to an LDAP attribute of the uploading user. The user can then view the status records in Web Center.

Alternatives to starting a handler program include:

- A watch service which is notified on changes of the upload folder and initiates appropriate actions.
- Manually checking for uploaded files.

1.1. Pre-Configured Variants

DirX Identity and Web Center come with three pre-configured file upload variants. Each

variant has an identifier (fileUpload1, fileUpload2, fileUpload3) which serves to associate configuration data with the corresponding variant.

You can easily customize the pre-configured variants. For example, you can configure all three variants to implement file uploads with different handler programs. Additional variants are supported but are more complex to configure.

1.2. Sample Use Cases

The standard Web Center application uses the pre-configured file upload variants to demonstrate different file upload use cases:

- Uploading a single file which is then stored on the server (fileUpload1).
- Uploading multiple files which are then stored on the server (fileUpload2).
- Uploading multiple files which are stored on the server and processed by a sample program. The Java program generates upload state records which can be displayed in Web Center as a list (fileUpload3).

The folder *install_path*/web/webCenter-***technical_domain***/samples/fileUpload* contains the files and folders used to run the sample program. These items are:

- **inputFiles** – some sample input files for the program. Subfolder **fileUpload1** contains some Excel files to test the first upload variant, which by default accepts excel files only. The input files in subfolders **fileUpload2** and **fileUpload3** match the configuration of the upload state form (displaying the status records) of the corresponding variant. But note that only variant fileUpload3 is pre-configured to write status records.
- **java** – the program's classes and source code.
- **work** – the program's working directory used to store uploaded files and for the program's standard output and error file.

1.3. Notation Conventions

In this document, we use shortcuts for some frequently referenced folders:

- *install_path* – the DirX Identity installation folder.
- *SAMPLES_FOLDER* – the file upload samples folder *install_path*/web/webCenter-*domain*/samples/fileUpload*.
- *TOMCAT_HOME* – the root folder of the Tomcat installation into which the Web Center application is deployed.

1.4. Related Documentation

The following documents provide additional details about the concepts and procedures referenced in this use case document:

- *DirX Identity Web Center Reference*, chapters "Configuration" and "User Interface Configuration".

- *DirX Identity User Interfaces Guide*, especially the chapters on Web Center.
- *DirX Identity Provisioning Administration Guide*, chapter "Managing Policies".

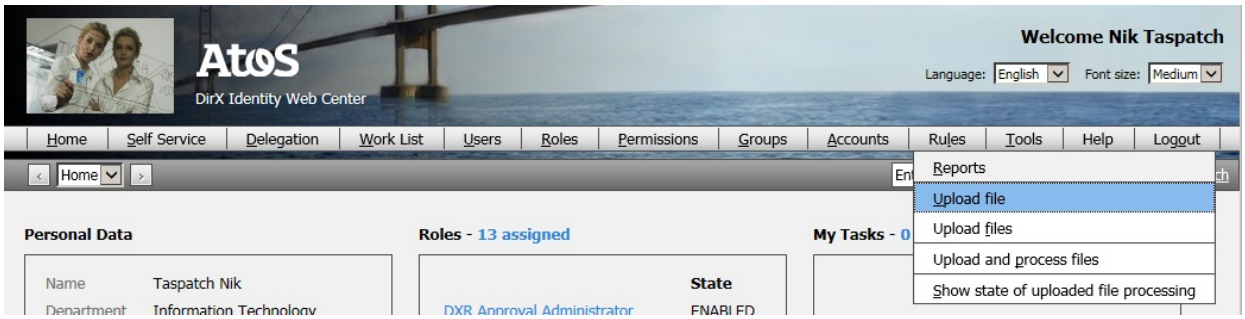
2. User Interface

This chapter gives a brief overview of the sample upload functionality as integrated into the standard Web Center application for demonstration purposes.

2.1. Menu Items

The Tools menu in the main menu bar includes four items related to file upload:

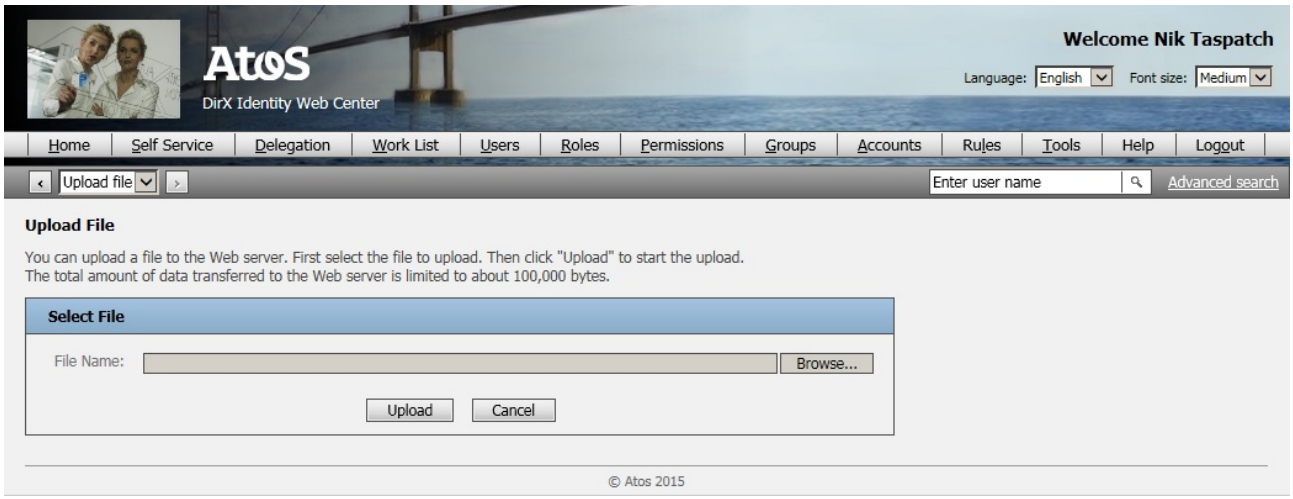
- **Upload file** – lets you select a file, uploads it to the server and stores it on the server (variant fileUpload1.) By default, only Excel files (.xls, .xlsx) are accepted.
- **Upload files** – lets you select multiple files, uploads them to the server and stores them on the server (variant fileUpload2.) By default, only text files (.txt) are accepted.
- **Upload and process files** – lets you select multiple files, uploads them to the server, stores them on the server and starts programs to process the files (variant fileUpload3.) By default, only text files (.txt) are accepted.
- **Show state of uploaded file processing** – displays status records that have been written by the handler programs (variant fileUpload3.)



The rest of this chapter describes these menu items in more detail.

2.2. Upload File

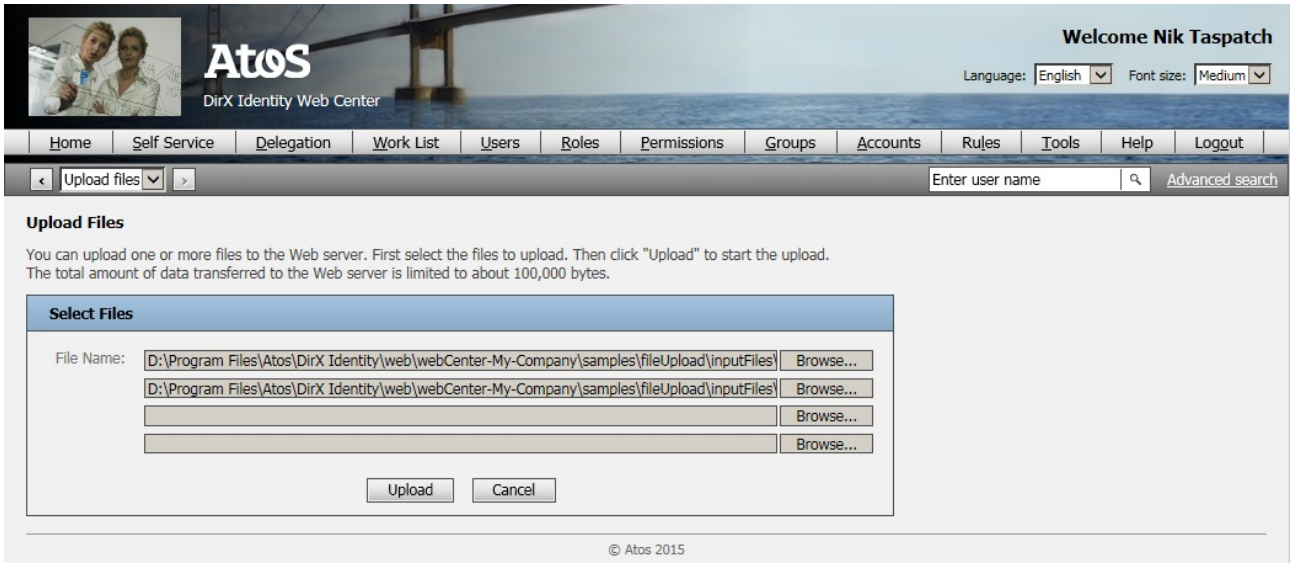
On selecting the menu item "Upload file", a page to upload a single file is displayed. The "Browse..." button lets you select a file. Clicking the "Upload" button starts the file transfer to the server.



The uploaded file is stored in the application's temporary folder (below *TOMCAT_HOME*/**work**). Its name is concatenated from the uploading user's DN and a time stamp.

2.3. Upload Files

On selecting the menu item "Upload and process files", a page to upload up to four files is displayed. The "Browse..." buttons let you select the files. Clicking the "Upload" button starts the file transfer to the server.

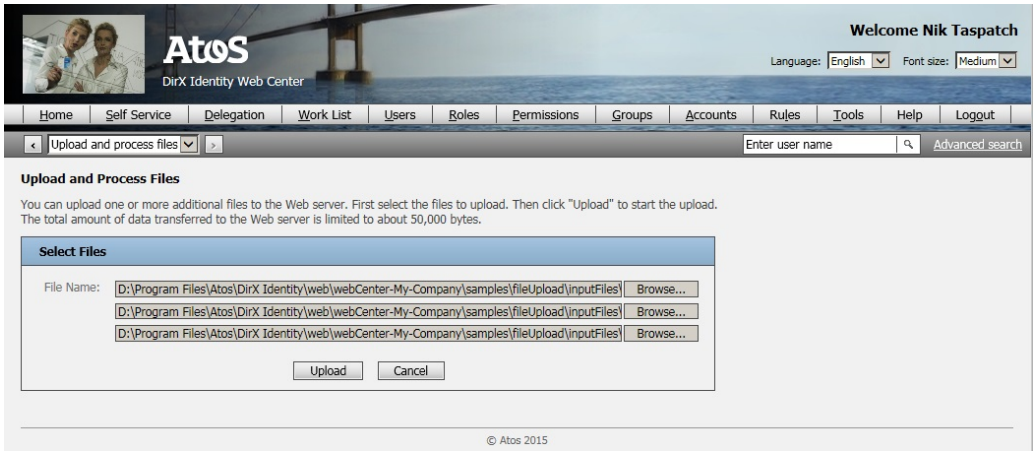


The uploaded files are stored in the application's temporary folder (below *TOMCAT_HOME*/**work**). Their names are concatenated from the uploading user's common name and the original file names.

2.4. Upload and Process Files

On selecting the menu item "Upload and process files", a page to upload up to three files is displayed. The "Browse..." buttons let you select the files. You'll find some appropriate sample input files in the folder *SAMPLES_FOLDER*/**inputFiles/fileUpload3**. Clicking the

"Upload" button starts the file transfer to the server.



The uploaded files are stored in the file upload samples folder (*SAMPLES_FOLDER***/work**). Their names are concatenated from the uploading user's common name and a time stamp.

For each uploaded file, the provided sample Java program is started to process the file. The program is located in the folder *SAMPLES_FOLDER***/java**. Its standard output and standard error are redirected to files in the folder *SAMPLES_FOLDER***/work**.

The sample program reads the input file line by line. Each line is expected to represent a state record, for example

```
${date}#${originalFileName}#false#30# 2# 2#0
```

The program performs some placeholder substitutions and then simply appends the records to the uploading user's LDAP attribute "dxrFileUploadState", for example:

```
fu3#20151222:102409#entries-30.txt#false#30# 2# 2#0
```

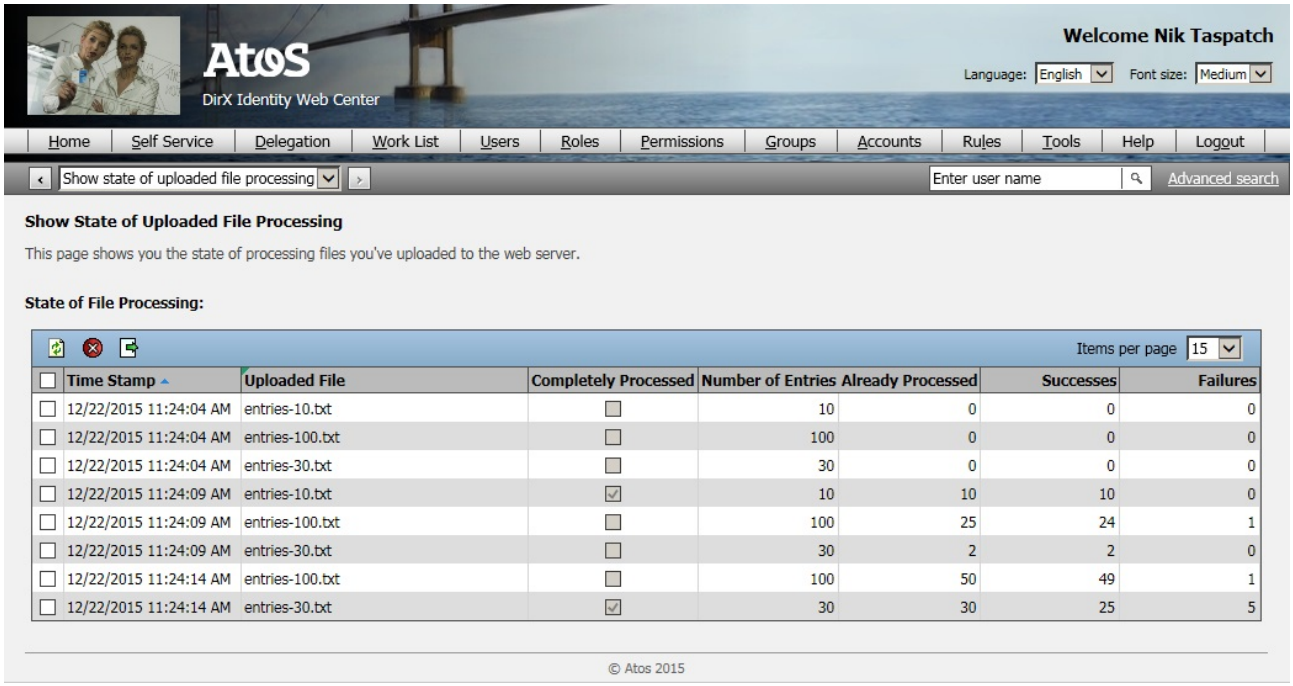
Note that the format of both the input files and the status records are specific to the provided sample program. Custom handler programs will usually have different input files and write differently-formatted status records.

The sample state records all start with **fu3#**. This notation serves to distinguish them from state records written by other handler programs. The time stamps ensure that the records are unique and they allow for sorting the records by their creation date.

2.5. Show State of Uploaded File Processing

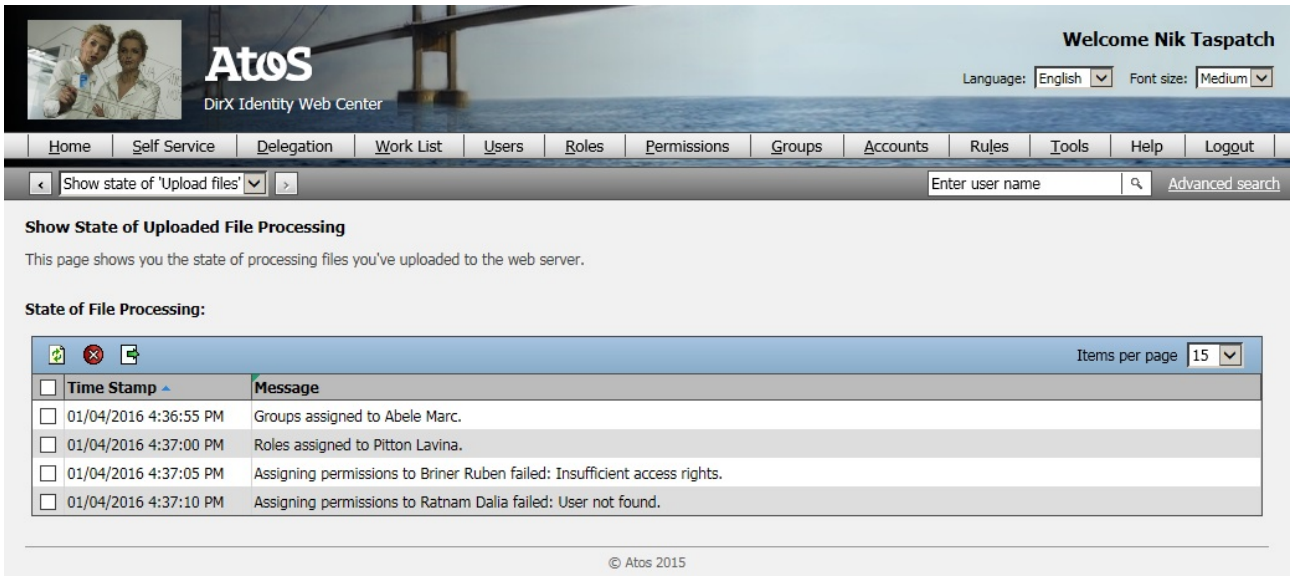
The menu item "Show State of Uploaded File Processing" lists the status records for the logged-in user written by the sample program. The list is configured to extract the different components from each record and to display them in separate columns.

You can refresh, export or delete the list or delete selected entries from the list.



The list columns depend heavily on the format of the status records written. Custom status records will usually be displayed in quite different lists. The list configuration must therefore be carefully adapted to the specific use case.

A simpler status record list might just display the time stamp and a message:



3. Configuration

This chapter describes how the sample upload functionality is configured into the standard Web Center application for demonstration purposes.

3.1. Menu Items

The file upload menu items must be added to a menu in the main menu bar. You could create a separate menu for them. In the Web Center standard application, however, we added them to the "Tools" menu.

There should be no need to customize the menu definitions unless you want to add the items to a different menu or your application supports more than three upload variants.

You'll find the menu extensions in the file **WEB-INF/config/identity/menu-defs.xml**.

3.1.1. Main Menu Bar

We extended the definition of the "Tools" menu with the file upload items:

```
<!-- Tools menu -->
<definition name=".menuTools" extends=".menu">
<put name="accessPolicyKey" value="Reports"/>
<put name="selectors"       value="ps,pwdId"/>
<put name="msgPrefix"       value="tools"/>
<put name="items"
     value="reports:/runReport.do;
            fileUpload1:/fileUpload1Select.do;
            showFileUpload1State,fileUpload1:
                /showFileUpload1State.do::fileUploadState1;
            fileUpload2:/fileUpload2Select.do;
            showFileUpload2State,fileUpload2:
                /showFileUpload2State.do::fileUploadState2;
            fileUpload3:/fileUpload3Select.do;
            showFileUpload3State,fileUpload3:
                /showFileUpload3State.do::fileUploadState3"/>
</definition>
```

The definition adds two items to the menu for each upload variant. The first one displays the page to select the file or files to upload:

```
fileUpload1:/fileUpload1Select.do;
```

The second item displays the list of upload state records:

```
showFileUpload1State,fileUpload1:
                /showFileUpload1State.do::fileUploadState1;
```

The second item is ignored if the corresponding upload variant is not configured to write status records.

A menu item is visible to a user only if there is a menu access policy granting him access to the item. Note that the menu key for the access policy is "Reports", while the menu items are "fileUpload1", "fileUpload2" and "fileUpload3", respectively.

3.1.2. Context Menu for the Upload State List

We added a new context menu for the list of upload state records for each upload variant:

```
<definition name=".contextMenuFileUpload3State">
    <put name="accessPolicyKey" value="Reports"/>
    <put name="msgPrefix"       value="ctx.objects"/>
    <put name="items"
         value="ms;delete,fileUpload3:confirm.fileUploadStates.yes;
                list;refreshList,fileUpload3;export,fileUpload3"/>
</definition>
```

The context menu includes functions to delete the selected states, to refresh the list and to export the list in HTML format (which is, for example, suitable for importing into Excel).

Deletion must be confirmed, provided that a text is assigned to the key "ctx.objects.delete.confirm.fileUploadStates" in the message file **text.properties**. The pre-selected button in the confirmation box is "yes". To pre-select "no", remove the ".yes" appendix from "confirm.fileUploadStates.yes".

3.1.3. Toolbar for State List

We added a new context menu for the list of upload state records for each upload variant:

```
<definition name=".toolbarFileUpload3State">
    <put name="menu"  value=".contextMenuFileUpload3State"/>
    <put name="items" value=
"refreshList::0;deleteList:delete;export"/>
</definition>
```

The toolbar displays three icons for refreshing the list, for deleting all state entries in the list,

and for exporting the list in HTML format. The icon to refresh the list reacts on clicks even if the list is empty (since the minimum number of entries is set to 0).

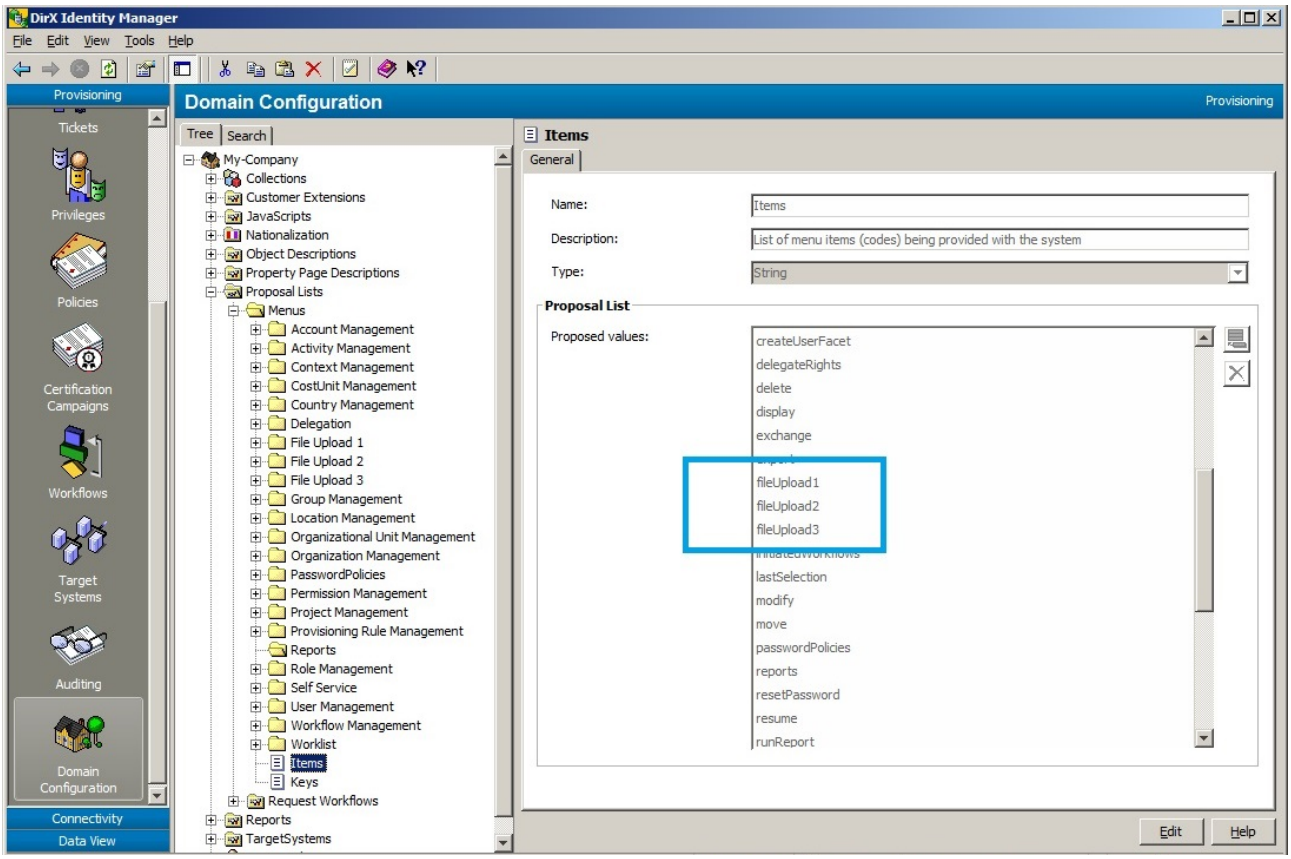
Deletion must be confirmed, provided that a text is assigned to the key "ctx.objects.deleteList.confirm.fileUploadStates" in the message file **text.properties**. The pre-selected button in the confirmation box is the same as for the context menu.

3.2. Menu Definitions

We extended the menu definitions to support the three file upload variants out-of-the-box.

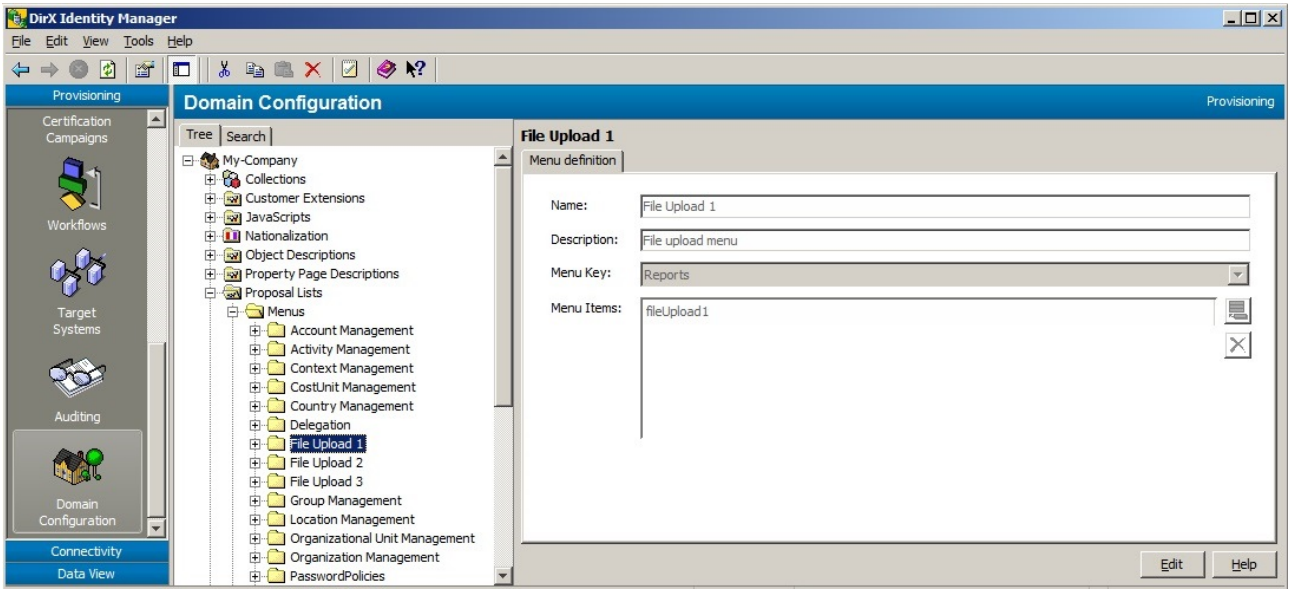
3.2.1. Menu Items Proposal List

The list includes the items fileUpload1, fileUpload2 and fileUpload3.



3.2.2. Menu Definition

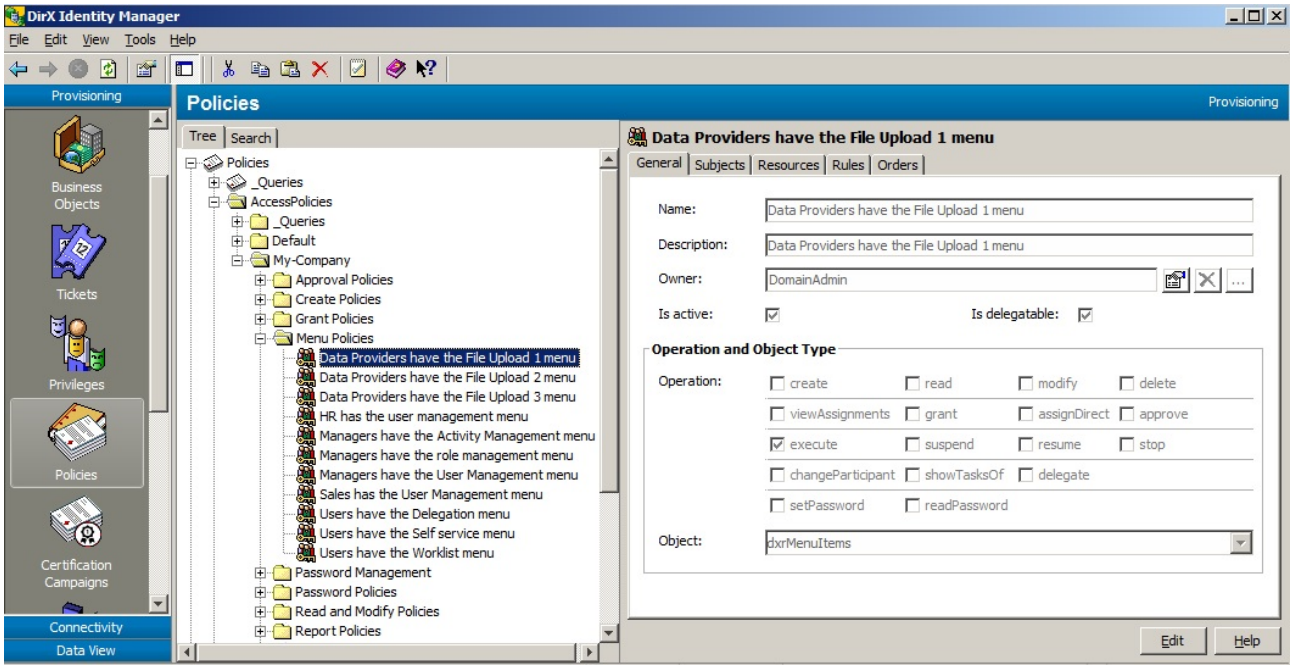
There is a menu definition for each file upload variant:



In the section "Main Menu Bar" above, we added the file upload menu items to the "Tools" menu. The access policy key for the "Tools" menu is "Reports". That's why the menu key in the menu definition is "Reports".

3.3. Access Policies

Menu access policies define which users have access to which file upload variant. The pre-configured policies in the DirX Identity sample domain allow users in the DirXmetaRole group "DataProviders" to access all three variants.



Tab "Subjects" assigns the group "DataProviders" to the field "Group of Persons".

Tab "Resources" assigns the corresponding menu definition (from the previous section) to the field "Resources".

For a customer domain, create corresponding access policies for the file upload variants you want to use and then assign the authorized users as subjects.

3.4. Settings

This section describes the configuration parameters for uploading, storing, and processing files in file **WEB-INF/config/webCenter-FileUpload.properties**. Note that the configuration is per file upload variant. The parameters of a specific variant start with the respective file upload identifier (fileUpload1, fileUpload2, etc.). All parameters support expressions unless otherwise stated.

3.4.1. Specific Variables

The following variables are made available for use in the settings:

- **docBase** – the Web application's document root folder.
- **environmentVariables** – the map of environment variables of the Tomcat process.
- **systemProperties** – the map of system properties of the Tomcat process.
- **applicationScope** – the map of application-scoped variables of the Web application.

Sample use cases:

- **applicationScope['javax.servlet.context.tempdir']** – the temporary folder for the Web Center application; the folder is provided by Tomcat.
- **systemProperties['java.home']** – the home folder of the Java installation used by Tomcat.
- **environmentVariables.DIRXIDENTITY_INST_PATH** – the DirX Identity installation folder (if any).
- **environmentVariables['catalina.home']** – the root folder of the Tomcat installation.

3.4.2. Uploading Files

This section describes some general configuration parameters for uploading files.

- **acceptedFiles** – the file types to accept for upload, for example, "application/vnd.ms-excel,.xls". The file types are evaluated by the browsers when a user selects a file. The types are supported by Chrome, Firefox, Edge and Internet Explorer. The browsers, however, do not check whether a selected file is really of the requested type. Therefore, this setting is for end user convenience only, but not to ensure that the server only gets uploaded files of the requested types.
- **accessPolicy.item** – The menu access policy item, usually one of "fileUpload1", "fileUpload2" or "fileUpload3". See the section "Menu Items" in the chapter "Configuration" for details.
- **accessPolicy.key** – the menu access policy key for the file upload variant, usually "Reports". See the section "Menu Items" in the chapter "Configuration" for details.
- **baseFolder** – a folder name which is referenced via ${baseFolder} in other settings. Just

for convenience.

- **handlerPage** – **t**he JSP to handle uploaded files. Web Center is currently shipped with two default handlers:
- **/WEB-INF/jsp/controller/tools/fileUpload/storeFiles.jsp** – stores uploaded files in a folder on the server**.**
- **/WEB-INF/jsp/controller/tools/fileUpload/storeFilesAndStartProgram.jsp** – stores uploaded files in a folder on the server and starts a handler program to process the files.
- **lockFileName** – the full path name of the lock file name. If you want to block uploads for some period of time, just create the lock file. The lock is released by removing the file. Note that uploads already in progress when creating the lock file are not affected.
- **maxContentLength** – the maximum number of bytes to be transferred to the Web server in a single request (⇐ 2147483647). The amount is the total size of the uploaded files, plus some additional bytes used by the HTTP transfer mechanism.
- **messageSection** – the name of the relevant message section in files **text.properties**. The standard section is "fileUpload.single" for uploads supporting just a single file, and "fileUpload.multiple" for uploads supporting multiple files.
- **numberOfFiles** – the maximum number of files that can be selected on the upload page. The default is 1.

3.4.3. Checking Uploaded Files

To prevent malicious content to be uploaded to the server, the server should check the content before accepting it.

3.4.3.1. Built-In Checks

Along with the file content, the browser sends the file name and the file content type to the server. This data is not reliable and can be easily be spoofed, but anyhow, we can check them first:

- **acceptedContentTypes** - the comma-separated list of accepted content types. If the list is empty, the check is skipped.
- **acceptedExtensions** - the comma-separated list of accepted file name extensions. If the list is empty, the check is skipped.

The next check tries to determine the mime type of the uploaded file by looking at its content. The check is based on the Apache TIKA library and does a fairly good job though it’s not perfect:

- **acceptedMimeTypes** - the comma-separated list of mime types accepted by the built-in mime type detector. If the list is empty, the check is skipped.

To find out which mime type the detector returns for a file, enable debug logging for Web Center and search in the log output for "Mime type of file".

3.4.3.2. Custom Checks

You can plug in your own content type detector. The detector must be a Java class extending the standard Java class "java.nio.file.spi.FileTypeDetector". The class must implement the abstract method "public String probeContentType(Path path)" which returns the detected content type, or null if the file type is not recognized. Note that the detector must be based on the file content only, not on the file name.

Then create a jar file containing the class and a folder named "META-INF/services". The folder should contain a file with name "java.nio.file.spi.FileTypeDetector" which lists the fully-qualified name of the custom class. See the Java API documentation of method "java.nio.file.Files.propbeContentType(Path path)" for details.

Finally, append the jar file to Tomcat's class path, for example via the Tomcat configuration program, or one of the batch files **setenv.bat** and **setenv.sh** in folder ***TOMCAT_HOME*/bin**.

Web Center provides the jar file and the Java source for a no-op custom file type detector in folder ***SAMPLES_FOLDER*/java**.

When a file is uploaded, the content type returned by the custom class is matched against configuration parameter

- **acceptedCustomContentTypes** - the comma-separated list of content types accepted by the custom file type detector. If the list is empty, the check is skipped.

3.4.3.3. Storing Files

This section comprises some configuration parameters for storing uploaded files in a folder on the server. They mainly define the upload file names, which are the names of the files the uploaded files are copied to on the server.

- **storeFiles.dateFormat** – the format of the time stamps to be included in upload file names (see class **java.text.SimpleDateFormat**).
- **storeFiles.fileNameExtension** – the upload file name extension.
- **storeFiles.fileNameTemplate** – the upload file name template. Special expressions are:
  - **${user.***attrName*} – the value of an LDAP attribute of the uploading user, like **user.dn** or **user.sn**.
  - **${date}** – a time stamp formatted as specified in the parameter **storeFiles.dateFormat**.
  - **${originalBaseName}** – the original (client-side) file's base name, without extension. For example, if the original file name is "c:\tmp\alpha.xls", the base name is "alpha".
  - **${originalExtension}** – the extension of the original (client-side) file name, which is the part of the file name starting with the last dot. For example, if the original file name is "c:\tmp\alpha.xls", the extension is ".xls".
  - **${originalFileName}** – the original (client-side) file name, comprised of base name and extension. For example, if the original file name is "c:\tmp\alpha.xls", the file name is "alpha.xls".

    ${count} – a file counter per upload action. Starts with 0.

- **storeFiles.folder** – the folder to store uploaded files in. The folder must exist and be writable by Tomcat.
- **storeFiles.invalidFileNameChars** – characters which are invalid in file names. A regular expression (see class **java.util.regex.Pattern**).
- **storeFiles.overwrite** – if an upload file already exists, whether to overwrite the existing file or to reject the upload request.
- **storeFiles.replacementString** – the replacement string for invalid file name characters defined in **storeFiles.invalidFileNameChars**; must not contain any dollar ($) or backslash (\). Default is a dash: "-".
- **storeFiles.userAttributes** – the comma-separated list of user attributes used in **storeFiles.fileNameTemplate**, for example "dn", or "sn,givenName".

3.4.4. Processing Files

This section describes some configuration parameters for starting the program to process uploaded files.

- **startProgram.base64Prefix** – the prefix for arguments to be passed Base64-encoded to the handler program. The default prefix is "base64:".
- **startProgram.programName** – the program name.
- **startProgram.programArgs** – the program arguments, separated by spaces. Special expressions are:
- **${loginDN}** – the uploading user's DN.
- **${language}** – the uploading user's language.
- **startProgram.workingDirectory** – the program's working directory.
- **startProgram.inputFileFolder** – the folder for the program's input file.
- **startProgram.inputFileNameTemplate** – the template for the program's input file.
- **startProgram.outputFileFolder** – the folder for the program's output file.
- **startProgram.outputFileNameTemplate** – the template for the program's output file.
- **startProgram.errorFileFolder** – the folder for the program's error file.
- **startProgram.errorFileNameTemplate** – the template for the program's error file.

3.5. Upload State Form

The form displaying the list of upload state values is defined in file **WEB-INF/config/tools/fileUpload***n*/**forms-config.xml**.

The form definition depends on the type of status records written by the program processing the uploaded files. We first show the definition of a simple form (fileUpload2), and then of a more complex form (fileUpload3). They correspond to the screenshots shown in the chapter "User Interface".

3.5.1. A Simple Form

The definition of a simple upload state form just displaying a time stamp and a message looks like this:

```
<form-bean name="fileUpload2StateForm">
  <form-property name="fileUploadState" y="+1" use="selectedItem"
        label="none" labelRenderer="empty" size="15"
        readonly="true"
        contextMenu=".contextMenuFileUpload2State:ms,list"
        valuePattern="fu2#\s*(\d{8}:\d{6})\s*#(.+)"
        type="com.siemens.webMgr.model.DirectoryEntryBean[]"
        width="100%" secondarySortColumns="$2">
    <data-property name="$1" type="java.util.Date"
        width="15%" label="fileUploadState.timeStamp"
        cellRenderer="sortableTimeWithSeconds"
        rendererProperties="format:yyyyMMdd:HHmmss"/>
    <data-property name="$2" type="java.lang.String"
        width="85%" label="fileUploadState.message"
        cellRenderer="toggleSelection" />
  </form-property>
  <form-property name="selectedItem" type="java.lang.String"
        value="-1" transient="true" visible="false"/>
</form-bean>
```

The value pattern is a regular expression (see the Java class **java.util.regex.Pattern**) for the state attribute values. State attribute values that don't match the expression are ignored.

The data property with the name "$n" references the n^{th} capturing group of a matching value. "$0" matches the entire value. Supported data property types are

- **java.lang.String** – string values.
- **java.lang.Boolean** – boolean values ("true", "false").
- **java.lang.Integer** – number values.
- **java.util.Date** – time stamps. The renderer property "format" defines the time stamp format as defined by the class **java.text.SimpleDateFormat**.

3.5.2. A Complex Form

The definition of the more complex sample file upload state form from variant fileUpload3 looks like this:

```
<form-bean name="fileUpload3StateForm">
```

```
  <form-property name="fileUploadState" y="+1" use="selectedItem"
        label="none" labelRenderer="empty" size="15"
        readonly="true"
        contextMenu=".contextMenuFileUpload3State:ms,list"
        valuePattern="sew#\s*(\d{8}:\d{6})\s*#\s*([^#]+)\s*#\s*
          ([^#]+)\s*#\s*(\d+)\s*#\s*(\d+)\s*#\s*(\d+)\s*#\s*(\d+)\s*"
        type="com.siemens.webMgr.model.DirectoryEntryBean[]"
        width="100%" secondarySortColumns="$1,$2,$5">
    <data-property name="$1" type="java.util.Date"
        width="15%" label="fileUploadState.timeStamp"
        cellRenderer="sortableTimeWithSeconds"
        rendererProperties="format:yyyyMMdd:HHmmss"/>
    <data-property name="$2" type="java.lang.String"
        width="25%" label="fileUploadState.fileName"
        cellRenderer="toggleSelection" />
     <data-property name="$3" type="java.lang.Boolean"
        width="12%" label="fileUploadState.finished"
        align="center"/>
    <data-property name="$4" type="java.lang.Integer"
        width="12%" label="fileUploadState.totalCount"
        align="right"/>
    <data-property name="$5" type="java.lang.Integer"
        width="12%" label="fileUploadState.processedCount"
        align="right"/>
    <data-property name="$6" type="java.lang.Integer"
        width="12%" label="fileUploadState.successCount"
        align="right"/>
    <data-property name="$7" type="java.lang.Integer"
        width="12%" label="fileUploadState.errorCount"
        align="right"/>
  </form-property>
  <form-property name="selectedItem" type="java.lang.String"
        value="-1" transient="true" visible="false"/>
</form-bean>
```

3.6. Nationalization

This section describes nationalization features.

3.6.1. Messages and Labels

Texts for messages and labels are defined as usual per supported language in the resource file **WEB-INF/classes/resources/languages/***language***/text.properties**.

Relevant are all message texts in the sections:

- fileUpload
- fileUpload*n*
- fileUploadState
- fileUpload*n*State
- tools.showFileUpload*n*
- tools.showFileUpload*n*State

and the two message texts with keys

- ctx.objects.delete.confirm.fileUploadStates
- ctx.objects.deleteList.confirm.fileUploadStates

3.6.2. Header Texts

The headers are stored per supported language in folder **WEB-INF/classes/resources/languages/***language***/tools/fileUpload***n*:

- **select.jsp** – The JSP switches between **selectSingleFile.jsp** and **selectMultipleFiles.jsp**. There should be no need to change this file.
- **selectMultipleFiles.jsp** – The header text for uploading multiple files. It's a JSP because it displays the configured maximum number of bytes to upload.
- **selectSingleFile.jsp** – The header text for uploading a single file. It's a JSP because it displays the configured maximum number of bytes to upload.
- **showState.html** – The header text for the page displaying the list of upload states.

3.7. Hints

This section provides hints on using Web Center file upload features and functions.

3.7.1. Upload Folder

The folder must exist and must be writeable by Tomcat.

Web Center does not clean up the folder.

If the folder is the temporary folder provided by Tomcat for the Web Center application, Tomcat may occasionally clean up the folder. This folder should therefore be taken as a storage folder just for demonstration purposes.

3.7.2. Handler Program

Web Center just starts the handler program. Web Center does not wait for the program to finish, nor does it check its progress or result. If the program crashes or stops prematurely, it is not restarted.

The program runs independently of Tomcat. Stopping or restarting Tomcat does not affect the program.

3.7.2.1. Argument Encoding

Arguments for a handler program may include special characters or characters from exotic character sets. To be able to pass such arguments safely, Web Center can be instructed to Base64-encode their values. The handler program must then decode the values.

Web Center encodes an argument value if it is prefixed with the Base64 identifier in the program's argument list defined in file **webCenter-FileUpload.properties**. The default Base64 identifier is "base64:". The prefix can be changed via configuration parameter **fileUpload*n***.startProgram.base64Prefix*. The actual Base64 prefix can also be passed to the program as an additional argument.

The encoded values will be prefixed with the Base64 identifier. For example, the value "dirx" is passed as "base64:ZGlyeA==" for Base64 arguments to the handler program.

Base64 encoding/decoding requires converting the argument values from Strings to byte arrays and vice versa. We use UTF-8 as the underlying character set for the conversions.

Java provides the utility class "java.util.Base64" to encode and decode strings. The following code snippet shows how to decode an argument value:

```
import java.nio.charset.StandardCharsets;
import java.util.Base64;

private static String decodeArgument(
               String arg, String base64Prefix) {
  String decodedArg = arg;
  if (arg != null && base64Prefix != null
                  && arg.startsWith(base64Prefix)) {
    String encodedArg = arg.substring(base64Prefix.length());
    byte[] dec = Base64.getDecoder().decode(encodedArg);
    decodedArg = new String(dec,StandardCharsets.UTF_8);
  }
  return decodedArg;
}
```

3.7.2.2. Status Entries

Status entries are stored as values of an LDAP attribute of the uploading user. Since attribute values must be unique, you will get an error when trying to add the same status record twice. Therefore you should take appropriate measures to prevent identical status records; for example, by including a sufficiently detailed time stamp.

3.7.3. Error Handling

If multiple files are uploaded in a single action, execution stops on encountering the first error while uploading or processing the files.

3.8. Sample Handler Programs

This section describes the sample handler programs provided with the Web Center file upload feature.

3.8.1. StatusEntryWriter

The program reads the content of the uploaded file line by line. If a line matches a status record template, a corresponding status record is written to the uploading user's LDAP entry. The program then waits for a configurable period of time before processing the next line.

3.8.1.1. Main Class

The main class is **net.atos.dirx.dxi.webCenter.fileUpload.StatusEntryWriter**.

3.8.1.2. Arguments

The handler program uses a UNIX-like argument style: –<argName> <argValue>

Supported arguments are:

- **base64Prefix** – the prefix for Base64-encoded arguments. Default: "base64:".
- **dateFormat** – the date format for replacing ${date} expressions in input files with the current time. Default: "yyyyMMdd:HHmmss".
- **host** – the host name or IP address of the LDAP server of the DirX Identity Provisioning database. Default: "localhost".
- **inputFile** – the name of the uploaded file. Required.
- **originalFileName** – the client-side name of the original file. It is used for replacing ${originalFileName} expressions in the input file.
- **port** – the port number of the LDAP server of the DirX Identity Provisioning database. Default: 389.
- **prefix** – the prefix for status records. Default: "sew#".
- **pwd** – the password for the uploading user. Default: "dirx".
- **sleep** – the time (in seconds) to sleep before writing the next status record. Default: 3.

- **statusAttribute** – the LDAP attribute name of the user attribute to append the status records to. Default: "dxrFileUploadState".
- **user** – the uploading user's DN. Required.

3.8.2. DummyFileProcessor

The program writes its arguments and working folder to standard out and then just waits for some time before finishing.

3.8.2.1. Main Class

The main class is **net.atos.dirx.dxi.webCenter.fileUpload.DummyFileProcessor**.

3.8.2.2. Arguments

The handler program uses a UNIX-like argument style: –<argName> <argValue>

Supported arguments are:

- **sleep** – The time (in seconds) to sleep before finishing. Default: 10.

DirX Product Suite

The DirX product suite provides the basis for fully integrated identity and access management; it includes the following products, which can be ordered separately.

DirX Identity

DirX Identity provides a comprehensive, process-driven, customizable, cloud-enabled, scalable, and highly available identity management solution for businesses and organizations. It provides overarching, risk-based identity and access governance functionality seamlessly integrated with automated provisioning. Functionality includes lifecycle management for users and roles, cross-platform and rule-based real-time provisioning, web-based self-service functions for users, delegated administration, request workflows, access certification, password management, metadirectory as well as auditing and reporting functionality.

DirX Directory

DirX Directory provides a standards-compliant, high-performance, highly available, highly reliable, highly scalable, and secure LDAP and X.500 Directory Server and LDAP Proxy with very high linear scalability. DirX Directory can serve as an identity store for employees, customers, partners, subscribers, and other IoT entities. It can also serve as a provisioning, access management and metadirectory repository, to provide a single point of access to the information within disparate and heterogeneous directories available in an enterprise network or cloud environment for user management and provisioning.

DirX Access

DirX Access is a comprehensive, cloud-ready, scalable, and highly available access management solution providing policy- and risk-based authentication, authorization based on XACML and federation for Web applications and services. DirX Access delivers single sign-on, versatile authentication including FIDO, identity federation based on SAML, OAuth and OpenID Connect, just-in-time provisioning, entitlement management and policy enforcement for applications and services in the cloud or on-premises.

DirX Audit

DirX Audit provides auditors, security compliance officers and audit administrators with analytical insight and transparency for identity and access. Based on historical identity data and recorded events from the identity and access management processes, DirX Audit allows answering the “what, when, where, who and why” questions of user access and entitlements. DirX Audit features historical views and reports on identity data, a graphical dashboard with drill-down into individual events, an analysis view for filtering, evaluating, correlating, and reviewing of identity-related events and job management for report generation.

For more information: support.dirx.solutions/about

Legal remarks

On the account of certain regional limitations of sales rights and service availability, we cannot guarantee that all products included in this document are available through the Eviden sales organization worldwide. Availability and packaging may vary by country and is subject to change without prior notice. Some/All of the features and products described herein may not be available locally. The information in this document contains general technical descriptions of specifications and options as well as standard and optional features which do not always have to be present in individual cases. Eviden reserves the right to modify the design, packaging, specifications and options described herein without prior notice. Please contact your local Eviden sales representative for the most current information. Note: Any technical data contained in this document may vary within defined tolerances. Original images always lose a certain amount of detail when reproduced.