EVIDEN

Identity and Access Management

DirX Identity

WebCenter Reference

Version 8.10.14, Edition March 2026

Table of Contents

Preface

The *DirX Identity Web Center Reference* is designed to obtain reference information about the DirX Identity Web Center. It consists of the following sections:

- Chapter 1 gives a brief overview of the files installed with Web Center.
- Chapter 2 provides information about Web application characteristics.
- Chapter 3 provides an architectural description of DirX Identity Web Center.
- Chapter 4 provides information about Web Center configuration.
- Chapter 5 provides information about Web Center user interface configuration.
- Chapter 6 provides information about Identity API used by Web Center.
- Chapter 7 provides information about Web Center utility classes.
- Chapter 8 provides information about Web Center tag library.
- Chapter 9 provides information about how to secure Web Center applications.
- Chapter 10 provides information about single sign-on configuration.
- Chapter 11 provides useful links to learn more about technologies used in Web Center.

DirX Identity Documentation Set

*Version 8.10.14 | Build 1858 | Date 2026-03-26 *

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. Installed Files

This chapter gives a brief overview of the files installed with Web Center. Some of the files are described in greater detail in other chapters of this manual.

After installation, the files are located in the file:

- *install_path*/**web/webManager.zip** (Windows) or
- *install_path*/**web/webManager.tar** (UNIX).

The DirX Identity Configuration program unpacks the archive to folder

- *install_path*/**web/webCenter-***domain*

with the following subfolders:

- **api** – API samples and documentation
- **endorsed** – Files used by the DirX Identity Configurator to configure Web Center and deploy the Web Center applications into Tomcat.
- **samples** – Sample files for Web Center's file upload feature.
- **shared** – A Java library used for single sign-on via SAP logon tickets.
- **tools** – The Log Viewer tool. The tool is described in the "Utilities" chapter of the *DirX Identity User Interfaces Guide*.
- **webCenter** – Web application files

The DirX Identity Configurator unpacks the archive again on each run, thereby overriding possible changes to files in the **webCenter-***domain* folder.

1.1. API Samples and Documentation

The folder **api** includes two subfolders:

- **docs-api** – The Javadoc specification of the API classes.
- **samples** – Java source files showing how to use the API, and a small Windows batch file-based environment to set up and perform some API tests.See file **samples/readme.html** for details.

1.2. Configurator Files

The files in folder **endorsed** are used by the DirX Identity Configurator to

- Complete the installation of Web Center; for example, by copying Java libraries from other DirX Identity components to the **WEB-INF/lib** folder.
- Integrate Web Center with other DirX Identity components, for example by replacing placeholders in **web.xml** and setting passwords in **password.properties**.
- Extend Tomcat's Java classpath with jar files delivered with DirX Identity.

1.3. Context Descriptors

The context descriptors serve to integrate the Web Center applications into Tomcat and to define some configuration parameters that vary with the application.

- **selfService.xml** – The context descriptor of the Self Service application.
- **webCenter.xml** – The context descriptor of the main Web Center application.

  The DirX Identity Configurator replaces placeholders in the files (@DOC_PATH@) with the correct document root folder for the respective application.It also copies file **webCenter.xml** to Tomcat's context descriptor folder and renames it to **webCenter-***domain***.xml**, thereby deploying the main Web Center application into Tomcat.The Self Service application, on the other hand, is not deployed by default.

1.4. Web Application Files

The **webCenter** folder serves as document root for the web applications.It contains two subfolders and a JSP file:

- **applets** – Contains the applet for digital signing.
- **resources** – Static files to be downloaded by clients (that is, browsers).
- **WEB-INF** – Files that are required for request processing on the server but should not be downloaded by clients.
- **index.jsp** – The application start page.

All files below the document root folder (besides the ones in subfolder **WEB-INF**) are subject to be downloaded by clients.Therefore, the folder should contain only those files that are intended to be served to clients.In particular, you should never put any sensitive files into the folder.

1.4.1. Resources Folder

The resource files are located in two subfolders:

- **build** – Static files to be downloaded by clients. The URLs Web Center clients use to download the static resources vary with the Web Center version. This ensures that the static resources are downloaded anew after product upgrades, without having the end users to click refresh or empty their browser caches.
- **config** – The configuration file **config.js**.
- **html** – The source file of the customized confirmation box: **confirm.html**.
- **messages** – Message files.
- **de** – German message file **message.js**.
- **en** – English message file **message.js**.
- **scripts** – Javascript source files

- **main** – A number of Javascript files that are downloaded by clients only as part of the file **main.js**, together with batch files to generate **main.js** anew after changes to the source files. Bundling several script files into a single large file reduces the number of HTTP requests required to load or reload a Web Center page.
- **styles** – CSS style sheets.
- **large** – The style sheet with large font sizes: **styles.css**.
- **medium** – The style sheet with medium font sizes: **styles.css**.
- **small** – The style sheet with small font sizes: **styles.css**.
- **help** – Help files.
- **de** – German help files. For each standard Web Center application, the (empty) help file itself (like WebCenterHelp.pdf) and a corresponding HTML wrapper (like WebCenterHelp.html).
- **en** – English help files. For each standard Web Center application, the (empty) help file itself (like WebCenterHelp.pdf) and a corresponding HTML wrapper (like WebCenterHelp.html).
- **images** – Image files.
- **bg** – CSS sprites for background images. CSS sprites reduce the number of HTTP requests needed to download background images (or to check for newer versions). The file **sprites.html** gives an overview of the contents of the sprites.
- **logos** – Company logos.
- Other image files.

1.4.2. WEB-INF Folder

The application server (Tomcat) ensures that files in this folder are never served to a client. The folder contains the subfolders:

- **classes**
- **log4j.properties** – The log4j configuration file.
- **META-INF**
- **jax-ws-catalog.xml** – A configuration file for the request workflow client component of Web Center.
- ***.dtd** – Document type definitions for some XML configuration files.
- ***.tld** – Tag library descriptors.
- **resources**
- **languages** – English and German message files (**text.properties**) and static HTML and JSP fragments.
- **config** – Configuration files.
- **bo** – Struts configuration files for business objects.
- **certifications** – Struts configuration files for access certifications.

- **delegation** – Struts configuration files for old delegations.
- **delegations** – Struts configuration files for new delegations.
- **functionalUsers** – Struts configuration files for functional users.
- **identity** – Struts configuration files with basic definitions and for login actions, and the main menu definition file.
- **imports** – Struts form bean fragments which are imported by other form beans.
- **links** – Struts configuration files for object selections.
- **personas** – Struts configuration files for personas.
- **policies** – Struts configuration files for password policies and provisioning rules.
- **privileges** – Struts configuration files for roles, permissions, groups and accounts.
- **tools** – Struts configuration files for reports.
- **userFacets** – Struts configuration files for user facets.
- **users** – Struts configuration files for users.
- **workflows** – Struts configuration files for request workflows and tasks.

In addition to that, the folder contains the following files:

- **converters.properties** – Web Center user interface configuration file.
- **defaultRenderer.properties** – Web Center user interface configuration file.
- **filters-config.xml** – Configuration file for search filters with DN syntax attributes.
- **messages.properties** – A mapping of some low-level error messages to message keys; a last resort to localize internal messages and display them with a more user-friendly text.
- **objects-config.xml** – Web Center user interface configuration file.
- **paths.properties** – Some lists of Struts actions and JSPs.
- **renderers-config.xml** – Web Center user interface configuration file.
- **request.properties** – A list of variables to be removed from request scope before processing an action.
- **session.properties** –Lists of session attribute names used to initialize and clean-up sessions.
- **validation.xml** – Struts forms validation configuration file.
- **validator-rules.xml** – Struts forms validation configuration file.
- **webCenter.properties** – High-level configuration file to control some features of Web Center.
- **configPwd –** Configuration files variants for the Web Center for Password Management application.
- **identity** – Struts configuration files with basic definitions and for login actions, and the main menu definition file.
- **users** – Struts configuration files for users.

- **configSelf –** Configuration files variants for the Self-Service application.
- **identity** – Struts configuration files with basic definitions and for login actions, and the main menu definition file.
- **users** – Struts configuration files for users.
- **jsp** – Java Server Pages.
- **controller** – JSPs performing the application's business logic.
- **data** – JSPs generating XML formatted data responses to asynchronous HTTP requests.
- **view** – JSPs generating the user interface.
- **lib** – Java libraries for Web Center applications. Note that all jar files in the folder are automatically loaded at runtime. If you put a newer version of a jar file into the folder, remove the old one from this folder, but don't just rename it.
- **snippets** – HTML and Javascript code snippets for user interface controls.

In addition to that, the folder contains two files:

- **password.properties** – The passwords required to access other DirX Identity components, keystores and truststores. The file is changed by the DirX Identity Configuration program. Passwords entered in cleartext will be encrypted on application restart.
- **web.xml** – The application's deployment descriptor. The file is changed by the DirX Identity Configuration program.

2. Web Application Characteristics

This chapter provides information about Web application characteristics.

2.1. Tomcat and Java Versions

Web Center runs with Tomcat 9.0. Web Center runs with Java 11.

2.2. Installed Files in the Tomcat Folder

The (initial) DirX Identity Configurator copies some files to folders below Tomcat's home directory, represented in this description as *tomcat_home*.Note that the Web Center application itself is not copied to Tomcat.

Automatic integration into Tomcat requires that the DirX Identity Configurator has the appropriate access rights to create and delete files and folders below Tomcat's home directory, and to adjust file and folder permissions.If the access rights are insufficient, the automatic integration will fail.In this case, complete the integration by performing the steps listed below manually.

2.2.1. Tomcat 9.0

The context descriptor file *install_path***/web/webCenter-***domain***/webCenter.xml** is copied to the folder *tomcat_home***/conf/**[*enginename*]/[*hostname*] under the new name **webCenter-***domain***.xml**.The folder and its parent folder are created if they don't exist.

The Java archive *install_path***/web/webCenter-***domain***/endorsed/lib/dmxStorageURL.jar** is copied to the folder *tomcat_home***/dxilib**.The folder is created if not yet existing.

The Java classpath for Tomcat is extended with the jar files copied to *tomcat_home* **/dxilib**.This is a replacement for the endorsed mechanism used by earlier versions of DirX Identity.Java 11 no longer supports the endorsed mechanism.Make sure the Java system property "java.endorsed.dirs" is not set for Tomcat.

The work folder *tomcat_home***/work/**[*enginename*]/[*hostname*]**/webCenter-***domain* is deleted (if it exists.)

On UNIX, the files are created with permissions 644 and the folders with permissions 755. The files must at least be readable for the Tomcat service.For file **password.properties** see the corresponding section in the chapter "Security".

enginename and *hostname* vary with the Tomcat installation.The engine name is usually **Catalina** or **Standalone**, the host name **localhost**.

2.3. Tomcat Configuration

Web Center encodes any HTTP request and response data based on UTF-8 in order to support the full range of Unicode characters.This applies also to the values of HTTP GET

request parameters.Tomcat, however, decodes such values by default based on ISO-8859-1. Therefore, parameter values containing characters outside 7-bit ASCII are incorrectly passed to Web Center.

To fix the problem, open file *tomcat_home***/conf/server.xml** and add attribute “useBodyEncodingForURI” with value “true” to any connector in use (like HTTP, HTTPS, AJP), For example:

<Connector port="8080" protocol="HTTP/1.1"

useBodyEncodingForURI="true" ...

Note that Web Center sends most HTTP requests via POST.Since POST requests are not affected the problem may take some time to show up after installation.

2.4. Running Tomcat behind a Proxy or Load Balancer

Tomcat may run behind a proxy server (or a load balancer).The browsers send their HTTP requests to the proxy, and the proxy forwards them to Tomcat.The responses are sent by Tomcat to the proxy, and then forwarded from the proxy to the browsers.

If the browsers connect to the proxy via HTTPS while proxy and Tomcat are connected via HTTP, a problem with redirected requests might occur.

Tomcat generates redirect URLs starting with “http://<tomcat-host>:<tomcat-port>”.Some proxies replace the host but leave protocol and port untouched: “http://<proxy-host>:<tomcat-port>”.The browsers then follow the redirect using HTTP instead of HTTPS to the proxy, and also probably with the wrong port.

You can fix this by changing the HTTP connector in Tomcat’s configuration file *tomcat_home*/conf/server.xml*.Set the “scheme” to “https” and set attributes “proxyName” and “proxyPort” to the host name and port of the proxy server, respectively.

<Connector port="8080" protocol="HTTP/1.1" ...
scheme="https"
proxyName="<proxy-host>"
proxyPort="443"
/>

Setting the scheme to “https” does not mean that Tomcat and the proxy use HTTP over SSL.

In case the browsers connect to the proxy via HTTPS and the proxy to Tomcat via HTTP, you should also tell Tomcat to add the **secure**-flag to session-cookies.This tells the browsers to send session-cookies only over secure connections.By default, Tomcat does not set the **secure**-flag in such cases because it’s contacted by the proxy via an unsecured connection.You can enable the secure-flag in file **WEB-INF/web.xml**:

<session-config>

...
<cookie-config>
<secure>true</secure>
</cookie-config>
</session-config>

2.5. Deploying Web Center on a Remote Server

Web Center may run on a remote server other than the central DirX Identity server with the provisioning store.

You don’t have to install DirX Identity on the remote server.But doing so makes configuration a bit easier.In both cases, configuration involves steps on the central and the remote machine.

Remote Server with DirX Identity Installation

Central Server

- Install and configure DirX Identity (with or without Web Center) on the central server.
- Make sure that remote access to any server to be contacted by the remote Web Center is not prohibited by a firewall. This affects directory, message broker and request workflow server (See section Socket Ports below.)

Remote Server

- Install DirX Identity on the remote server. As components to install, select only “Provisioning directory schema and data” and “Web Center”.
- Configure DirX Identity on the remote server. As configuration options, select “Web Center Configuration” only.
- On the following pages, enter host, port etc. of the central directory server for provisioning, and the domain information. For Web Center, enter the local Tomcat’s installation directory and service name.
- If applicable, set up single sign-on as described in the User Interface Guide.

Remote Server without DirX Identity Installation

Central server

- Install DirX Identity.
- If you don’t have a Tomcat installation on the central server or don’t want to deploy Web Center into that installation, create a dummy Tomcat installation as described at the end of this section.
- Configure DirX Identity, thereby deploying Web Center into the (dummy) Tomcat installation on the central server. When deploying Web Center into a dummy Tomcat installation, do not enter a Tomcat service name and don’t check the flag to start the Tomcat service after installation.

- Make sure that remote access to any server to be contacted by the remote Web Center is not prohibited by a firewall. This affects directory, message broker and request workflow server (See section Socket Ports below.)
- Copy the **webCenter-***domain* folder from the central server's *install_path***/web** folder to any folder on the remote machine.

Remote server

- Copy the Java archives in folder **webCenter-***domain***/endorsed/lib** to folder *tomcat_home***/dxilib**. Create the folder if it does not exist.
- Extend Tomcat's Java classpath with the full path names of the jar files in the folder *tomcat_home***/dxilib**.
- Adjust the configuration parameters in file **webCenter-***domain***/webCenter/WEB-INF/web.xml**:
- **com.siemens.webMgr.ldap.host**
- **com.siemens.webMgr.ldap.port**
- **com.siemens.webMgr.ldap.ssl**
- **com.siemens.webMgr.requestworkflow.keystoreName** – If applicable; note that the placeholder **@DIRXIDENTITY_INST_PATH@** cannot be replaced at runtime since there is no installation.
- If applicable, set up SSL to the directory server, message server or Java-based Server as described in the *DirX Identity Connectivity Administration Guide*.
- If applicable, set up single sign-on as described in the User Interface Guide.
- Copy the context descriptor file **webCenter-*****domain***/webCenter.xml* to folder *tomcat_home***/conf/**[*enginename*]**/**[*hostname*] and rename the copy to **webCenter-*****domain*** (or any other name you want to use to access the Web Center application in a browser.) Create the folder if not yet existing. Open the file and change the **docBase** to the full path name of the **webCenter-***domain***/webCenter** folder.
- Make sure that all files and folders have the appropriate access rights. The files must be readable for Web Center, which means for the Tomcat server Web Center is deployed into. For file **password.properties** see the corresponding section in chapter "Security".
- If existing, delete folder *tomcat_home***/work/**[*enginename*]/[*hostname*]**/webCenter-***domain*.
- Restart Tomcat.

enginename and *hostname* vary with the Tomcat installation. The engine name is usually **Catalina** or **Standalone**, the host name **localhost**.

Dummy Tomcat Installation

A dummy Tomcat installation is any folder with the following subfolders:

- **bin**
- **dxilib**

- **conf**
- **conf/Catalina**
- **conf/Catalina/localhost**

2.6. Hints on Tomcat

- Tomcat is delivered with some sample applications.In a productive environment, most sample applications are not needed, and you should undeploy them by removing the respective files from folders *tomcat_home***/webapps** and *tomcat_home***/conf/**[*enginename*]/[*hostname*].
- On some systems, Tomcat may log error messages “SEVERE: Error filterStart**”** during start-up.These messages are caused by some of the sample applications and do not affect Web Center.
- For a slight but easy performance improvement, set the initialization parameter **trimSpaces** of the JSP servlet (with servlet name **jsp**) in *tomcat_home***/conf/web.xml** to “true”.
- When running under Tomcat on a UNIX or Linux system, Web Center pages may occasionally be broken due to a failed attempt to access an X display server.In Tomcat’s log file, you will find an error message like "Can’t connect to X11 window server using '0:0' as the value of the display".Since access to an X display server is not really required, we recommend setting the Java system property **java.awt.headless** to **true** on such systems; for example, in *tomcat_home***/bin/catalina.sh**: **JAVA_OPTS=-Djava.awt.headless=true**.

2.7. Peculiarities and Restrictions

This section provides information about peculiarities and restrictions.

Reloadability

Web Center is reloadable.If you reload (or stop and restart, or re-deploy) the application in the Tomcat Web Application Manager, Tomcat will log some messages about threads and ThreadLocals that might cause a memory leak but you can safely ignore them.

Undeployment

If you undeploy a Web Center application in the Tomcat Web Application Manager, its context descriptor file is deleted.

On changes to a context descriptor file, Tomcat re-deploys the respective application, which leads to the same warnings as mentioned above.

Session Persistence

Web Center sessions are not serializable.That means if you reload the application or restart Tomcat, all sessions get lost.

Tomcat, by default, attempts to persist sessions on application unload.This feature is disabled in Web Center's context descriptor **webCenter-*domain*.xml**.

Distributability

Since the sessions are not serializable they are also not distributable among several servers in a Tomcat cluster.You may run Web Center behind a load balancer with sticky sessions only.

2.8. Socket Connections

Web Center interacts with several servers via socket connections.It reads the servers' host names, ports and SSL flags from **web.xml** or the configuration database.

The host names must be appropriately specified.Names like "localhost" and "127.0.0.1" or simple host names will not work if Web Center and the server reside on different machines or in different domains.We recommend using fully qualified host names or IP addresses.

Also, if a server is located on a different host than Web Center, any intermediate firewall must grant Web Center access to the server port.

2.8.1. Directory Server for Provisioning

Host, port and SSL flag are taken from **web.xml**.The default ports are 389 for non-SSL connections and 636 for SSL-connections.

2.8.2. Directory Server for Connectivity

Host, port and SSL flag are taken from the provisioning directory. In the DirX Identity Manager, open view "Provisioning/Domain Configuration" and select tab "General" of the domain object.

The default ports are 389 for non-SSL connections and 636 for SSL-connections.

2.8.3. Message Broker

Web Center contacts the message broker for sending events like password change events.

Host, port and SSL flag are taken from the connectivity directory. In the DirX Identity Manager, open view "Connectivity/Expert View" and go to object "Configuration/Services/System/Message Broker *index*".

The default port is **61616** for non-SSL and **61617** for SSL connections.

In a high availability scenario, ensure access to primary and secondary message broker.

2.8.4. Java-based Server

Web Center contacts the Java-based Server for processing request workflows.

Host, port and SSL flag are taken from the Connectivity directory. In the DirX Identity Manager, open the view “Connectivity → Expert View”. For port and SSL flag, go to object “Configuration → Services → System → *Java-based Server name*”. For the host name, open the system object that is referenced on that page.

The default port is **40000** for both SSL and non-SSL connections.

In a high availability scenario, ensure access to primary and secondary Java-based Servers.

3. Web Center Architecture

DirX Identity Web Center is a Java-based Web application that must be deployed on a servlet container such as Apache Tomcat. Web Center uses the following technologies:

- The Jakarta Struts framework of the Apache Software Foundation for action processing (controller).
- The Jakarta Tiles framework to create the graphical representation (view).
- Java Server Pages (JSP) to handle the business logic (model) via the Identity API (Java).

The architecture is shown in the following figure:

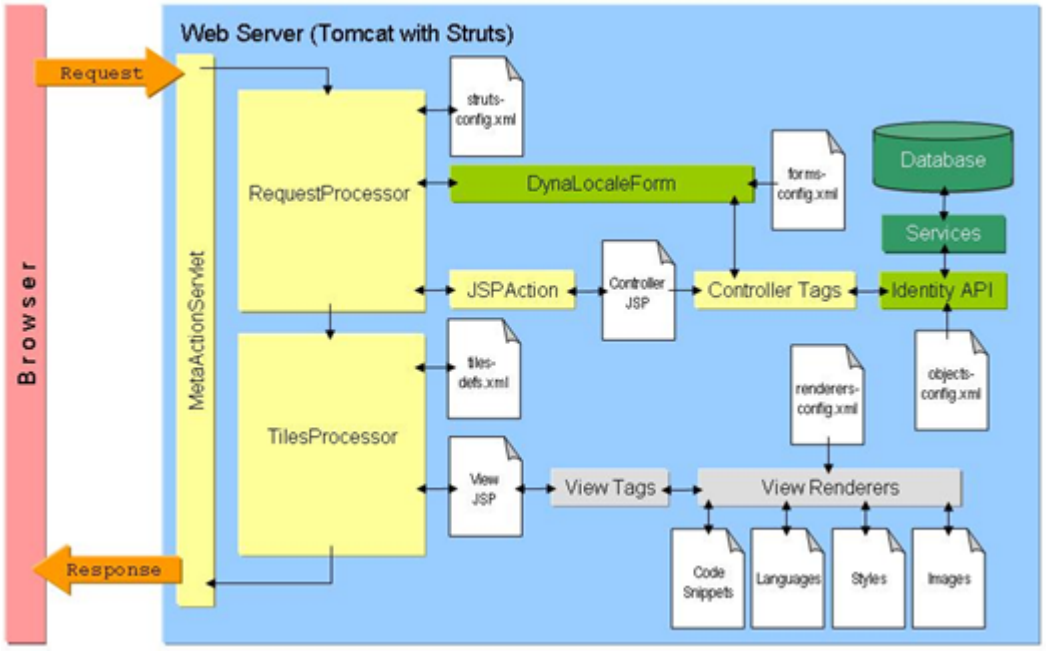


Figure 1. Web Center Architecture

3.1. Software Components

The application consists of the following structural components:

- Action servlet and action handler
- Controller JSP pages and tags
- Identity API
- View JSP pages and tags
- Renderers and resources
- Configuration files

The next sections describe these components in more detail.

3.1.1. Action Servlet

The action servlet is the main component of Web Center and represents the entire application to the servlet engine. When the servlet engine is started, the action servlet is the first Java class that is accessed, and performs all necessary initial configurations. When a client requests the top URL of the application, the request is passed to this class which forwards it to the request processor of the Struts framework. The request processor loads the necessary form component, calculates the action to be performed and forwards the request together with this information to Web Center's action handler.

3.1.2. Action Handler

The action handler is a Java class that performs the requested operation. Instead of implementing this type of Java class for each operation, it is implemented as a generic class, using a Java Server Page (JSP) that contains the proper action code for execution. When the JSP page is accessed for the first time, a special parser and compiler is used to convert the JSP page to a standard Java class file which is then reused for all subsequent requests for the same page as long as the content of the respective JSP file is not changed. This means that finally pure Java code is executed, but instead of having it bundled as a black-box Java archive, it can be modified according to the customer's needs.

3.1.3. Controller JSP Pages

The JSP pages used by the action-handler class are called controller JSP pages in contrast to the view JSP pages mainly used for the construction of the resulting HTML pages. The set of controller JSP pages located in **WEB-INF/jsp/controller** is divided into the following types of page:

- **Core pages**, which perform basic operations like login and logout, reading and storing data, and performing basic assignments. These pages are not typically customized but can be included into customized pages.
- **Task pages**, which are JSP pages for particular tasks like password change operations, delegation handling or user listing (under certain conditions).
- **Utility pages,** which are pure Java code pages coded to cover the majority of use cases. While JSP pages usually contain JSP tags and in some exceptions scripted pure Java code, this file category consists of pure Java code. We recommend that you do not change the implementation of these files.
- **Workflow pages,** which are special JSP pages required for all tasks associated with request workflows. These pages read workflow definitions and current tasks of the user, perform approvals, update activity states with the user's modifications and create signatures for the user's actions.

3.1.4. Controller JSP Tags

The content of the controller JSP pages usually consists of a logically-arranged set of JSP tags. While an extensive use of the Java Standard Tag Library (JSTL) is made, application-specific tags also exist and serve to hide the implementation of some basic operations like reading and storing form properties, performing the proper login or logout operation and

so on.

3.1.5. Identity API

The Identity API contains a set of interfaces for performing the proper identity management operations by the DirX Identity system. The interface classes are well documented and can be used to build add-on features not covered by the Web Center application. The interface classes hide implementation details of the DirX Identity service classes and provide an object model of solely basic Java types. The Web Center application in turn uses some own helper classes to convert these basic object types into convenient ones.

3.1.6. View JSP Pages

When the action handler has performed its request-specific task, it returns the request to the Struts request processor. To construct the resulting HTML pages, the application uses the Struts Tiles component. The Tiles framework allows page modularization and building pages out of global and specific components.

The view JSP pages located in **WEB-INF/jsp/view** do not represent complete result pages, but rather the components and building blocks of these pages. They are divided into two parts:

- **Form pages**
  The form pages folder contains the page layoutPage.jsp taken as a basis for all other pages except those appearing in pop-up windows. There are some other "higher integrated" pages used for special purposes (tab-sheet, assignment page and pop-up window).
- **Tiles pages**
  This set contains all page components used to construct tables, forms out of a form configuration list, menus, page headers and footers, and so on.

3.1.7. View JSP Tags

The view JSP tags are used for rendering complex user interface components like tables, tab-sheets, search panels, and tree controls. These tags are used in the view JSP pages in addition to the Java Standard Tag Library (JSTL). A view JSP tag together with an appropriate renderer set produces the final proper HTML code to be sent to the client that will display it in the browser window.

3.1.8. Renderers

The renderers are the face of the Web application.These are the modules that will produce the HTML code that is sent to the client and displayed in the Web browser window.A renderer receives its runtime data usually from a view JSP tag and uses predefined HTML or JavaScript code fragments called "snippets" that contain placeholders for the runtime data.The renderers format the data, replace the placeholders with runtime data and return the completed HTML or JavaScript code to the view tag that will add it to the response object.

3.1.9. Resources

The resource folders contain all necessary items to display a ready-made HTML page, like code snippets, text fragments and message strings in various languages as well as images and styles.Resources that should not be made directly accessible to clients lie in folders below **WEB-INF**.The other ones like images, styles and Javascript files reside in folder **resources** parallel to **WEB-INF**.

3.1.10. Configuration Files

The application requires a number of configuration files.Some are part of the Struts and Tiles framework (struts-config.xml, tiles-defs.xml), others are Web Center proprietary and define objects, forms and renderers.See the chapter “Web Center Configuration” for further details.

3.2. Typical Request Flow

When a request is sent from a client to the Web Center application, the servlet engine prepares and forwards it to the action servlet.

The action servlet processes the request using the Struts request processor.Struts transfers the request form data into the action form objects, selects the action based on the URL and the action mapping in the Struts configuration file and invokes the action handler.The action mapping contains the name of the form, the parameter variable (which is a set of name-value pairs here) and a set of possible forward definitions.A forward could be either another action URL or a Tiles definition.

The Web Center action handler copies any parameters defined in the action mapping of the Struts configuration to the request as its attributes, looks up the file with the controller JSP page to be used and passes it to the request dispatcher module.The dispatcher executes the controller JSP page which results in a series of controller JSP tag invocations.The controller JSP page functionality depends very much on the action.At a minimum, it calculates a forward name, stores it as a request attribute and returns to the action handler.Typically, it performs the necessary actions in the Identity domain, like searching for objects and storing attribute modifications.When the controller JSP page returns, the action handler looks up the requested forward and returns this item to the request processor.

This sequence is then repeated or, if the forward contains a Tiles definition, the Tiles processor is called to produce the HTTP response.The Tiles processor prepares the view JSP page to be rendered from usually nested definitions.Once the coding of the JSP page is completed, it is compiled and processed.

Processing the view JSP page leads to invocations of the view JSP tag library and in turn to the renderer set for formatting the final HTML page.The code of this page is passed to the servlet engine’s response object for sending a response to the client.

3.3. A Sample: List Users

Let’s study the typical flow for a sample.Assume the servlet receives an HTTP request with an action “/listSingleUser”.

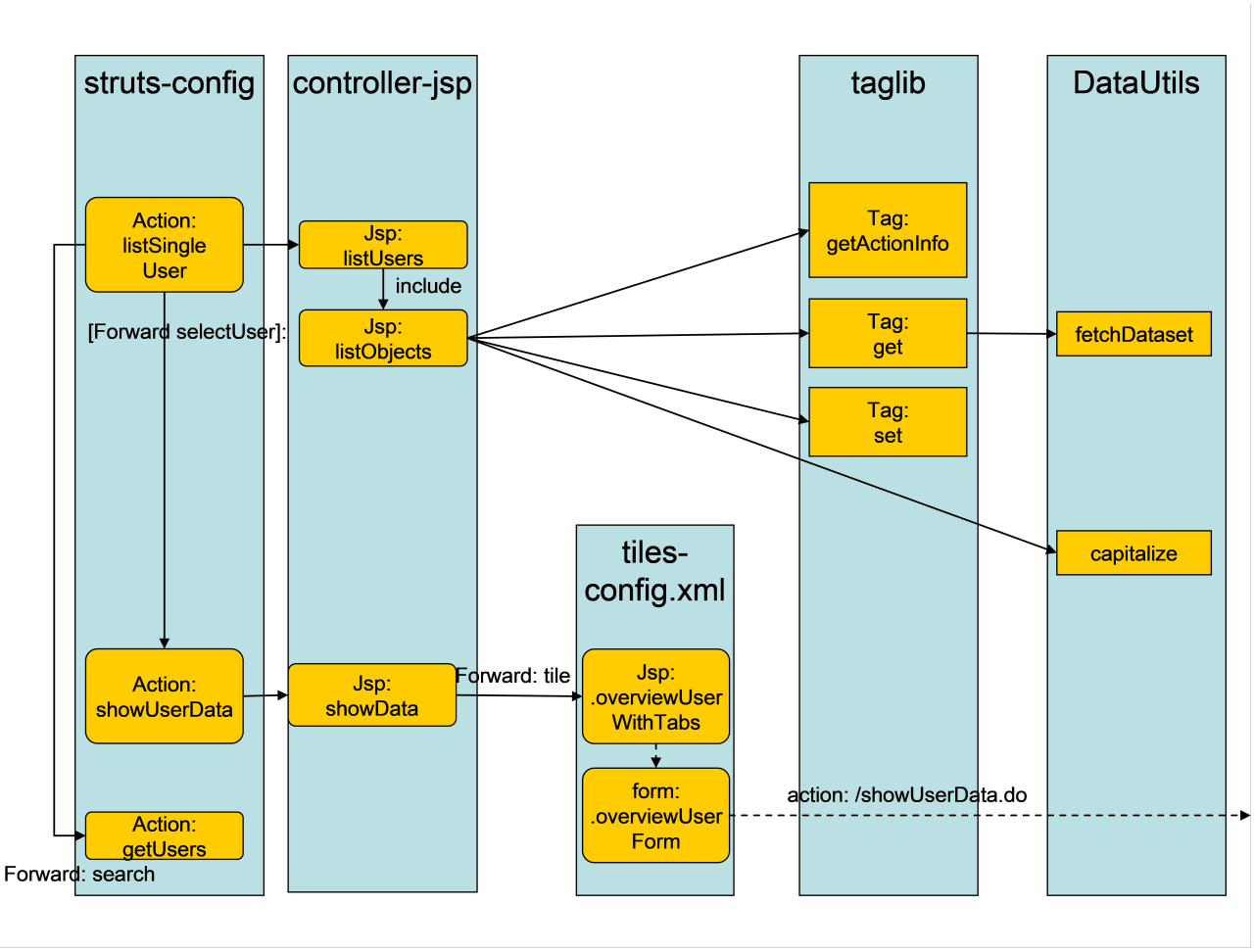
The following action mapping in the Struts configuration tells the Struts request processor to fill the form “userListForm” and to call the action handler “JSPAction”:

```
<action path="/listSingleUser"
        type="com.siemens.webMgr.controller.action.JSPAction"
        name="userListForm"
        parameter="jspPage:/WEB-
INF/jsp/controller/tasks/listUsers.jsp;
            elements:users;
            object:${sessionScope['com.siemens.webMgr.loginDN']};
 defaultSearchBase:cn=Users,${initParam['com.siemens.webMgr.ldap.base
DN']};
            defaultFilter:objectClass=dxrUser">
        <forward name="search" path="/getUsers.do"/>
        <forward name="selectUser" path="/showUserData.do"
redirect="true"/>
        <forward …/>
        <forward name="tile" path=".users"/>
</action>
```

The action handler takes the name of the controller JSP page “listUsers” from the parameter attribute, locates the JSP file and invokes the request dispatcher with the JSP page.

The following diagram gives an overview of the affected JSP pages, tag libraries, utility methods and the resulting action.

Example JSP Page Control Flow



The JSP page “listUsers” includes the page “listObjects”, which contains some tags; for example, “getActionInfo”, “get” and “set”. The tag implementation uses the Identity API and its associated Java utilities to read and update the Identity domain entries such as the user. For example, the “get” tag calls the method “fetchDataset” of the Java utility class “DataUtils” to search for objects according the filter, base node and other attributes passed.

Let’s assume one user is selected. The JSP page returns the forward “selectUser”. The action handler returns this forward to the Struts request processor, which in turn looks for the appropriate action mapping and may find the following:

```
<action path="/showUserData"
    type="com.siemens.webMgr.controller.action.JSPAction"
    name="overviewUserForm"
    parameter="jspPage:/WEB-INF/jsp/controller/core/showData.jsp;
objectVar:selectedUser;
object:${sessionScope['com.siemens.webMgr.selectedUser']}">
    <forward name="tile" path=".overviewUserWithTabs"/>
    <forward name="tileContentOnly" path="..."/>
</action>
```

The Struts request handler re-calls the same Web Center action handler “JSPAction” with the form “overviewUserForm”. The action handler finds the controller JSP page “showData”, which is typically used for presenting entity objects and their attributes.

The default forward references a Tiles definition “.overviewUserWithTabs”. It is defined in the Tiles configuration file “tiles-defs.xml”, where we may find the following:

```
<definition name=".overviewUserWithTabs" extends=".base">
        <put name="formContent" value=".overviewUserForm"/>
        <put name="formContent2" value=".userPrivilegesTab"/>
        <put name="formContent3" value=".userBinaryTab"/>
        <put name="textFile" value="overview.html"/>
    </definition>
```

The page definition extends a basic page definition and references a form definition “.overviewUserForm” which may be defined as follows:

```
<definition name=".overviewUserForm"
path="/WEB-INF/jsp/view/tiles/form.jsp">
      <put name="action" value="/showUserData.do"/>
      <put name="styleClass" value="standardList"/>
</definition>
```

The Tiles processor aggregates the complete HTML page from the given “tiles”. The user attributes are presented using the generic JSP page “form.jsp”. It contains some Web Center view tags, which evaluate the context beans and build the HTML code.

The resulting HTML page is passed back to the servlet, which returns it to the client browser.

4. Web Center Configuration

Web Center configuration is divided into three parts:[.indexref]##

- Web application configuration files.

The files contain settings for integration into Tomcat and for accessing the provisioning database and the request workflow service, as well as the list of Struts configuration files.

- Context descriptors **webCenter-*domain*.xml** and **selfService-*domain*.xml**.
- Deployment descriptor **web.xml**.
- High-level configuration files.

The files contain settings expressed in terms of features. The settings are intended to work the same way in future releases even if the underlying implementation is changed.

- Server-side configuration file **webCenter.properties**.
- Client-side configuration file **config.js**.
- Low-level user interface configuration files.

The files contain settings expressed in low-level terms based on their current implementation in Struts. Due to possible implementation changes, they are not guaranteed to work the same way in future releases.

This chapter describes the Web application and high-level configuration files. The low-level user interface configuration files are described in chapter "User Interface Configuration".

4.1. Context Descriptor Files

Context descriptor files like **webCenter-*domain*.xml** and **selfService-*domain*.xml** are located in Tomcat's **conf/[enginename]/[hostname]** folder.They announce the Web applications to the servlet engine by the declaration of the installation directories and additionally hold the configuration paths for struts-config.xml, tiles-defs.xml, renderers-config.xml, and objects-config.xml.

The context path used to specify a web application in URLs is inferred from the application's context descriptor file name.

The file comprises the following sections:

- Context

Context

The Context section has the following parameters:

- **cookies** – Whether cookies are to be used for session identifier tracking. Set this parameter to **true** if you want cookies to be used if the client supports them. Should be

enabled for security reasons.

- **crossContext** – Whether access to other contexts is required. Set to **false** since access to other contexts is usually not required.
- **docBase** – The application's document root directory.
- **override** – Whether settings in this context override any corresponding settings in either the global or host default contexts.
- **privileged** – Whether access to container servlets is required. Set to **false** since access to container servlets is not required.
- **reloadable** – Whether to automatically reload the application on changes to classes in WEB-INF/lib and WEB-INF/classes.
- **useHttpOnly** – Whether to set the HttpOnly flag on session cookies to prevent client-side scripts from accessing the session ID. Should be enabled (**true**) for security reasons.
- **useNaming** – Whether to initialize a JNDI context. Set to **false** since JNDI is usually not used by Web Center applications.

Parameter

The Parameter section contains context parameters in addition to those defined in the deployment descriptor **web.xml** (see also the section "Deployment Descriptor web.xml"). Context parameters are used in Web Center to set the list of Struts configuration files. These parameters include:

- **override** - Whether a context parameter of the same name defined in the Web application deployment descriptor overrides the value specified here (**true**) or not (**false**). Default: **true**.
- **name** - Context parameter name.
- **value** - Context parameter value.

Manager

The Manager section contains the parameter:

- **pathname** - Name of the file in which session state will be preserved across application restarts, if possible.Leave this parameter empty since Web Center applications do not support session persistence.

4.2. Deployment Descriptor web.xml

The Web deployment descriptor **web.xml** in **WEB-INF** holds all configuration data that is specific to the particular installation.

On the one hand, it contains data required by Web Center to access and collaborate with other DirX Identity components, like the address of the provisioning store, the domain name, bind information, and single-sign-on settings.On the other hand, it contains the definition and configuration of several Web Center components like filters, servlets and listeners.

The deployment descriptor comprises the following sections:

- Context Parameters
- Filter Definitions and Mappings
- Listener Definitions
- Servlet Definitions and Mappings
- Session Configuration
- Welcome File List
- Error Handlers
- Security Settings

4.2.1. Context Parameters

Each context parameter has a name and a value. The parameter names are case-sensitive.

Note: Context parameters may be defined as well in the context descriptor file.

4.2.1.1. Configuration File Names

The configuration file names must be specified relative to the application's document root folder and with a leading slash. Most file names for the standard Web Center applications are set in the respective context descriptor files.

- **ObjectConfig.Path** – A comma-separated list of names of configuration files containing the object definitions for the application.
- **RendererConfig.Path** – A comma-separated list of names of configuration files containing the renderer definitions for the application.
- **Struts.DefaultFiles** – The names of Struts and Tiles configuration files that are always included if found in one of the folders listed in parameter **Struts.Path**.
- **Struts.Path** – A list of names of configuration folders and files containing the definitions of Struts actions, Struts form beans, Tiles components and Web Center menus. A detailed description is included in the context descriptor files.

4.2.1.2. Enabling UTF-8

Web Center sets the character set for HTTP requests to UTF-8 by default, while the default character set for HTTP is ISO-8859-1.

- **com.siemens.webMgr.utf8.enabled** – Whether to use character set UTF-8 (**true**) or not (**false**). Default: **true**.

4.2.1.3. Default Language Configuration

Web Center pages are usually presented to a user in a language calculated by the value of the HTTP request header accept-language, provided Web Center supports one of the languages requested in that header. If not, Web Center falls back to the default language

which is set in the following Struts context parameter:

- **javax.servlet.jsp.jstl.fmt.fallbackLocale** - The default locale. The value is required and must match one of the languages supported by Web Center.

Web Center is shipped with built-in support for English (locale: en) and German (locale: de) but can be customized for other languages.

4.2.1.4. Directory Access Configuration

The context parameters to access the DirX Identity provisioning store are usually set by the DirX Identity configuration program. You can change the relevant parameters at any time.

- **com.siemens.webMgr.ldap.host** - The name or IP address of the directory server for provisioning.
- **com.siemens.webMgr.ldap.ssl** – Whether to access the directory server via a secured SSL connection (**true**) or not (**false**). Default: **false**.
- **com.siemens.webMgr.ldap.port** - The LDAP port of the directory server. Default: **389**.
- **com.siemens.webMgr.ldap.baseDN** - The DirX Identity provisioning domain (for example, **cn=My-Company**).
- **com.siemens.webMgr.ldap.user** - The technical user used to bind to the DirX Identity provisioning store (for example, **cn=DomainAdmin,cn=My-Company**). The technical user must have the appropriate directory access rights to perform all operations on behalf of the users logged in to Web Center. The Web Center users' access rights are calculated by DirX Identity access policies, not by directory access controls. The technical user's password must be added to Web Center's password file with identifier ldap.
- **com.siemens.webMgr.ldap.anyone** - The technical user used to calculate the DirX Identity access rights during user self-registrations (for example, **cn=ANYONE, cn=My-Company**). The technical user must have the appropriate DirX Identity access policies to perform all operations on behalf of the users registering but should not have any further access rights. The technical user's password must be added to Web Center's password file with identifier ANYONE.
  The directory operations required for user self-registrations are performed by the technical user defined in the previous section (com.siemens.webMgr.ldap.user). That user must therefore have the appropriate directory access rights.

4.2.1.5. Request Workflow Configuration

Web Center establishes a SOAP connection to the DirX Identity Java-based Server in order to perform request workflow tasks like creating new objects, displaying the work list or approving privilege assignments. The related configuration parameters are:

- **com.siemens.webMgr.requestworkflow.updateTimeout** – The maximum time for which new workflow requests are delayed by workflow engine calculations. Some requests require the workflow engine to recalculate the state of the affected workflow. New requests for the workflow are delayed until the calculation is finished. The update timeout defines the maximum delay time. If the calculation is not finished within the

specified period of time, the new request is cancelled. Unit: Milliseconds. Default: **0**.

Web Center usually forwards the current user's login credentials to the Java-based Server in order to authenticate the user to the Java-based Server. If you operate Web Center in single sign-on mode, however, the user's password is unknown to Web Center, and cannot be used to authenticate the user to the Java-based Server. In this case, Web Center authenticates itself to the server with its private key, and forwards only the name of the current user. The Java-based Server verifies that it has a valid certificate of the presented private key, and if so, accepts the username without further authentication.

Thus, in case of single sign-on, you must generate a private key for Web Center and put a corresponding certificate into the Java-based Server's truststore (*install_path***/ids-j-***domain* **-S***n***/private/webcenter-truststore**). We recommend using the same key and keystore as for SSL with client authentication from Web Center to the Java-based Server: For details, see the section "Establishing Secure Connections with SSL" in the chapter "Managing the Connectivity System" in the *DirX Identity Connectivity Administration Guide*. The following context parameters give Web Center the location of the keystore and the alias for the private key:

- **com.siemens.webMgr.requestworkflow.keystoreName** – The location of the keystore containing the private key of Web Center. If Tomcat and the Java-based Server reside on the same machine, the keystore should be *install_path***/ids-j-***domain***-S** *n***/private/webcenter-keystore-***alias*. If the servers are not co-located, you must copy the keystore to the machine hosting Tomcat.
- **com.siemens.webMgr.requestworkflow.keyAlias** – The alias for the private key in the keystore. Default: **WebCenter**.

The passwords required to access the keystore and the private key must be added to Web Center's password file with identifier webcenterKeystore and webcenterKey, respectively.

4.2.1.6. Client Request Configuration

A Web Center page may contain one or more tabs to download information not displayed by default, like privileges, photos, and certificates on a user summary page. On clicking a tab, the client sends an asynchronous request for the additional information to the server and inserts the response into the current page. If the server fails to respond within a given period of time, the client cancels the request.

- **com.siemens.webMgr.asyncRequest.timeout –** Specifies the timeout in seconds for the client to wait for a response. (JSP defined) Default: **30**.

4.2.1.7. Logging Configuration

For development purposes, Web Center provides a logging component printing detailed information about the processed requests to Tomcat's console or stdout log file, respectively. As of version 8.10, the logging component is used by

- The Web Center classes and filters.
- The DirXweb for JSP classes and filters.
- The Log tag <ctrl:log> and <view:log> used in JSPs.

This means that the log configuration now applies to all these classes, and they write into the same log file.

The logging facility is configured by the following parameters:

- **com.siemens.webMgr.log.level** – Specifies the log level:
- **-2** - Use log4j; see the parameter **com.siemens.webMgr.log.log4jConfigFile**.
- **-1** - OFF: Disables logs.
- **0** - SEVERE: Prints only severe errors (default).
- **1** - INFO: Prints additional information.
- **2** - FINEST: Prints detailed information.
- **com.siemens.webMgr.log.hideValuePattern** – Prevents possibly security-compromising variable values stored in request, session or application scope from being logged in clear text. The pattern value is interpreted as a regular expression matching the names of the variables to be hidden. The value ".*Date", for example, matches any name ending with date or Date. The pattern is applied in addition to the hard-coded pattern ".*Password|dxrChallengeResponse" preventing clear text logging of passwords and challenge-response values. For details on pattern syntax, google for the documentation of Java class **java.util.regex.Pattern**.
- **com.siemens.webMgr.log.dateFormat** – Web Center adds a time stamp to loggings from some low-level components that would otherwise be written without a timestamp. You can specify the time stamp format here. See the documentation of Java class **java.text.SimpleDateFormat** for details on the syntax.
- **com.siemens.webMgr.log.log4jConfigFile** – If you've set the log level to "-2" to activate log4j logging, you can configure log4j as usual in a properties file. By default, log4j takes file **WEB_INF/classes/log4j.properties**. If you run two Web applications from the same application folder, they work with the same log4j configuration and therefore write into the same log files. Since this setup may cause issues (for example, with log4j's RollingFileAppender), you can define a different configuration file per application. Note that you need to define this parameter in the context descriptor files of your applications since the deployment descriptor **web.xml** is also shared.

4.2.1.8. Online Debug Configuration

Web Center supports online debug information and an online LDAP trace. The output is displayed at the bottom of the regular Web Center pages. You can activate the trace by adding a parameter to any Web Center URL (for example, in your browser's address bar) with the name debug and a combination of one or more of the following characters as the value:

Character	Description
a	Print application-scoped attributes
e	Print system environment
r	Print request information, request headers and request-scoped attributes

Character	Description
s	Print session-scoped attributes
t	Print LDAP operation trace
o	Switch debug output off

The URL:

http://*yourTomcatServer:yourTomcatPort***/webCenter?debug=rs**

for example, activates request and session debug output. The current debug option is saved in session-scope and is therefore valid throughout the session or until explicitly overridden by a subsequent request.

- **com.siemens.webMgr.debug.enabled** – Enables (**true**) or disables (**false**) online debug info. Default: **false**.
- **com.siemens.webMgr.debug.trace.enabled** – Enables (**true**) or disables (**false**) the online LDAP trace. If used in multi-user mode, the traces of simultaneous working users are usually mixed up. Default: **false**.
- **com.siemens.webMgr.debug.trace.keywords** – A comma separated list of keywords for the LDAP trace. The supported keywords are **ldap** and, for a more detailed trace, **ldapconnection**. Default: **ldap**.

You should not enable the trace in a productive environment for security reasons. If enabled, every user can send a corresponding URL and see possibly security-compromising information on his or her screen.

When developing a customized application, you must provide a debug output area and check for the debug request parameter to use this facility. The output area for the standard Web Center applications is defined in **/WEB-INF/jsp/view/forms/layoutPage.jsp**, the debug request parameter is evaluated in **/WEB-INF/jsp/controller/utils/checkDebug.jsp** which is included from **/WEB-INF/jsp/controller/utils/checkSession.jsp**.

4.2.1.9. Session Monitor Configuration

Web Center includes a tool to monitor session and main memory usage. The tool writes a log entry to a file each time a Web Center session is created or destroyed. The logs are written to a number of files in a round-robin fashion.

Here is a sample log entry:

```
11:03:00 BBC30248DE6EADBC5E08E9A49D8ACDB8 4     - 00:02 [3,4]
[36/12/24]    [36/24]
```

It includes the following data:

Field		Value/Unit/Format
The current time		hh:mm:ss
The session ID		
The session number; the first session gets number 1, the next number 2 and so on		
An operation code	Session creation	+
	Session deletion	-
	Session was still open when Tomcat was stopped	o
Total session life time		hh:mm
Number of currently open sessions		
The maximum number of open sessions since Tomcat startup		
Total memory		Megabytes
Free memory		Megabytes
Used memory		Megabytes
Maximum total memory since Tomcat startup		Megabytes
Maximum used memory since Tomcat startup		Megabytes

The log files can be viewed with any text editor. The related configuration options are:

- **com.siemens.webMgr.sessionMonitor.enabled** – Enables (**true**) or disables (**false**) the session monitor. Default: **false**.
- **com.siemens.webMgr.sessionMonitor.fileName** – The log file name pattern, for example **c:/temp/SessionTrace-%d.txt**. The respective file number replaces the placeholder %d at runtime. The directory for the files must exist and must be writable by the Tomcat process. The file name pattern must be different for each deployed application. Default: *application_directory***/WEB-INF/SessionTrace-%d.txt**.
- **com.siemens.webMgr.sessionMonitor.linesPerFile** – The number of log entries to write to each file before switching to the next one. Default: **10000**.
- **com.siemens.webMgr.sessionMonitor.numFiles** – The maximum number of log files. Default: **10**.

4.2.1.10. Single Sign-On Configuration

For the various context parameters related to single sign-on see the chapters on single sign-on in the DirX Identity manuals.

4.2.2. Filter Definitions and Mappings

Filters are Java classes that perform tasks on a request before it is processed by the Struts controller (the MetaActionServlet), or after it has been processed by the controller.

A filter definition defines a filter with its name, the implementing Java class, and initialization parameters.

A filter mapping defines which requests a filter is applied to. Multiple filters may apply to the same request, in which case their execution order is determined by the order of matching filter mappings: from top to bottom before request processing, and in reverse order after request processing.

The following table lists the filters supplied with Web Center and the resources they are applied to. If several filters apply to the same resource, their filter mapping order must be from top to bottom. Before processing a Struts action (*.do), for example, the RequestFilter must run first, the CSRF filter next, then, if applicable, the SSOHeaderFilter, and the SessionFilter last.

The ClickjackingFilter and the MethodsFilter perform some general security checks. They should be invoked for each request before the actual request processing starts.

Filter	**Mapped Resource URIs**
ClickjackingFilter	/*
MethodFilter	/*
RequestFilter	*.do *.jsp /saveFile
BinaryRequestFilter	/binaryReader/*
CSRF Filter	*.do *.jsp /saveFile
SSOHeaderFilter	*.do *.jsp /saveFile
SessionFilter	*.do *.jsp
AddHeaderFilterForDownloads	/binaryReader/*

Filter	Mapped Resource URIs
AddHeaderFilterForStaticResources	/resources/* /resource/*
ExtAuthFilter	/login.do /login.jsp

4.2.2.1. ClickjackingFilter

The ClickjackingFilter is a DirXweb for JSP filter. It prevents clickjacking against Web Center pages by sending the proper HTTP response header **X-Frame-Options** that instruct the browser to not allow framing from other domains. For details on clickjacking see https://www.owasp.org/index.php/Clickjacking.

4.2.2.1.1. Filter Name

An arbitrary name for the ClickjackingFilter. The name is used in the filter mappings section.

4.2.2.1.2. Initialization Parameters

- **X-Frame-Options-Header –** The HTTP protocol versions the header should be applied to, and HTTP header name and value. There should be no need to change the parameter.
- ***:X-Frame-Options:SAMEORIGIN** – Apply the header to all HTTP versions (*****). Header name is **X-Frame-Options**, value is **SAMEORIGIN**.

4.2.2.1.3. Filter Mappings

The filter should be applied to each request:

- **/* –** Any URI.

4.2.2.2. MethodFilter

The MethodFilter is a DirXweb for JSP filter. It checks the HTTP method of each incoming request. GET and POST requests are always allowed. HEAD, OPTIONS and TRACE requests are optionally allowed. All other requests like PUT and DELETE requests are rejected. The filter also processes OPTIONS and TRACE requests directly instead of forwarding them down the processing chain.

4.2.2.2.1. Filter Name

An arbitrary name for the MethodFilter. The name is used in the filter mappings section.

4.2.2.2.2. Initialization Parameters

- **AllowedMethods –** The optionally allowed method names, separated by commas. Names other than **HEAD**, **OPTIONS** and **TRACE** are ignored. None of the optional

methods is allowed by default.

4.2.2.2.3. Filter Mappings

The filter should be applied to each request:

- **/* –** Any URI.

4.2.2.3. RequestFilter

The RequestFilter is a DirXweb for JSP filter. In the context of Web Center applications, it is used to synchronize requests for the same session, to properly decode request parameters, and to disable an annoying feature of the JavaServer pages standard tag library (JSTL).

4.2.2.3.1. Filter Name

An arbitrary name for the RequestFilter. The name is used in the filter mappings section.

4.2.2.3.2. Initialization Parameters

RequestFilter initialization parameters include:

- **ApplicableMethods** – A comma-separated list of HTTP methods this filter is applied to. Requests with other HTTP methods are just forwarded to the next handler in the processing chain. Specify **ALL** to apply the filter to all requests. Default: **GET,POST,HEAD**.
- **DecodeRequestParameters** – Whether the filter should decode request parameters (**true**) or leave the task to the servlet engine (**false**). Should be set to **false**. Default: **false**.
- **RequestSyncEnabled** – Whether to synchronize requests for the same session (**true**) or not (**false**). Note that the requests must be synchronized by some method since some session resources are not thread-safe. Simultaneous access to the same session may lead to unpredictable results. Therefore, we strongly recommend setting this parameter to **true**. Default: **true**.
- **RequestEncoding** – Character encoding of HTTP request parameters. Must be supplied with the character set used by the Web Center application, which is **utf-8** for the standard Web Center applications. Deprecated. Use context parameter **com.siemens.webMgr.utf8.enabled** instead.
- **IgnoreLocale** – Whether to prevent the JSTL formatting tags from resetting the character set in the HTTP response header Content-Type according to the client's preferred language. Note that the tags do not take utf-8 applications into account. For example, if the preferred language is English, the tags change the character set from utf-8 to iso-8859-1. Set this flag to **true** for Web Center applications using the utf-8 character set. Default: **true**.
- **URLRewritingEnabled** – Whether to allow tracking session IDs via URL rewriting (**true**) or not (**false**). URL rewriting is considered a security risk and should be turned off. Default: **false**.
- **SSLRequired** – Whether HTTPS is required to access the application (**true**) or not (**false**). If SSL is required, insecure requests are redirected to the URL defined in parameter **SSLRedirectURL**. If no redirect URL is specified, the request is rejected. Default: **false**.

- **SSLRedirectURL** – The URL to redirect insecure request to if SSL is required. Note that the URL and any parameters of the original request get lost and are not available when processing the redirected request. Default: **none**.
- **AcceptedCrossOrigins** – The origin servers which to accept cross-origin requests from. Separate the servers by commas. Each origin is evaluated as a case-insensitive regular expression (java.util.regex). For example, to accept the origins "https://alpha.gamma.com:8443" and "http://beta.gamma.com:8080" specify **https://alpha\.gamma\.com:8443**, **http://beta\.gamma\.com:8080**.

4.2.2.3.3. Filter Mappings

In order to let the application server know which requests are to be processed by the filter, the URI of any applicable Web Center request must be mapped to the filter. For the applications provided with Web Center, map the filter to the URIs

- ***.do** – Struts actions
- ***.jsp** – Java Server Pages
- **/saveFile** – The save file servlet

4.2.2.4. BinaryRequestFilter

The BinaryRequestFilter is identical to the RequestFilter but might be configured in a different way to meet the requirements of the BinaryReader servlet.

4.2.2.4.1. Filter Name

An arbitrary name for the BinaryRequestFilter. The name is used in the filter mappings section.

4.2.2.4.2. Initialization Parameters

For the parameters see the RequestFilter section above.

4.2.2.4.3. Filter Mappings

Map the filter to URI

- **/binaryReader/*** – The BinaryReader servlet

4.2.2.5. SessionFilter

The SessionFilter is a DirXweb for JSP filter. In the context of Web Center applications, it is used to perform some checks on incoming requests; for example, if the session requested by the client is still valid, or, in case of user self-registration, if the requested URI is allowed in that context. The checks are not hard-coded in the filter. Instead, the filter includes a JSP that performs the checks. When developing a customized Web Center application, you may provide your own JSP that performs modified or additional checks according to your requirements.

4.2.2.5.1. Filter Name

An arbitrary name for the SessionFilter. The name is used in the filter mappings section.

4.2.2.5.2. Initialization Parameters

SessionFilter initialization parameters include:

- **ApplicableMethods** – A comma-separated list of HTTP methods this filter is applied to. Requests with other HTTP methods are just forwarded to the next handler in the processing chain. Specify **ALL** to apply the filter to all requests. Leave the default for Web Center applications. Default: **GET,POST,HEAD**.
- **IncludePage** – The name of the JSP performing the checks. Specify the name relative to the application context. The page for the Web Center default applications is **/WEB-INF/jsp/controller/utils/checkSession.jsp**. An alternative page with the name **checkSessionWithLogging.jsp** writes additional logs which might be useful for analyzing login issues; for example, in single sign-on scenarios.
- **RestartPage** – The URI to forward the request to in case a check fails. Specify the URI relative to the application context. The restart page of the Web Center default applications is /restart.do.
- **RestartMode** – How to forward the request to the restart page. The supported modes are:
- **include** – By invoking the Java Servlet API method RequestDispatcher.include (default).
- **forward** – By invoking the Java Servlet API method RequestDispatcher.forward.
- **redirect** – Via HTTP redirection.

  For Web Center applications, use **forward**.

4.2.2.5.3. Filter Mappings

In order to let the application server know which requests are to be processed by the filter, the URI of any applicable Web Center request must be mapped to the filter. For the applications provided with Web Center, map the filter to the URIs

- ***.do** – Struts actions
- ***.jsp** – Java Server Pages

4.2.2.6. AddHeaderFilterForDownloads and AddHeaderFilterForStaticResources

The filters are differently configured instances of the same DirXweb for JSP filter. They are used to specify expiry dates for static resources like Javascript files and images, or for binary attributes like photos and certificates downloaded via the BinaryReader servlet. Expiry dates prevent the browser from unnecessarily checking for updated resources on each request.

4.2.2.6.1. Filter Name

An arbitrary name for the filter. The name is used in the filter mappings section.

4.2.2.6.2. Initialization Parameters

- **Expires-Header** – The expiry date of the downloaded resource. The date must be specified in number of seconds from download time (now), for example:
- **http/1.*:Expires:now+3600** – The resource expires 1 hour after download time.
- **http/1.*:Expires:now+86400** – The resource expires 1 day after download time.

4.2.2.6.3. Filter Mappings

Map the AddHeaderFilterForDownloads to URI

- **/binaryReader/*** – The BinaryReader servlet

Map the AddHeaderFilterForStaticResources to URI

- **/resources/*** – Static Web Center resources
- **/resource/*** – Static Web Center resources

4.2.2.7. CSRF Filter

The CSRF Filter attempts to prevent cross-site request forgery attacks. The filter is configured in file **/WEB-INF/config/webCenter.properties**.

4.2.2.7.1. Filter Name

An arbitrary name for the CSRF Filter. The name is used in the filter mappings section.

4.2.2.7.2. Initialization Parameters

None.

4.2.2.7.3. Filter Mappings

In order to let the application server know which requests are to be processed by the filter, the URI of any applicable Web Center request must be mapped to the filter. For the applications provided with Web Center, map the filter to the URIs

- ***.do** – Struts actions
- ***.jsp** – Java Server Pages
- **/saveFile** – The save file servlet

4.2.2.8. SSOHeaderFilter

For details on the SSOHeaderFilter, see the chapters on single sign-on in the DirX Identity User Interfaces Guide.

4.2.3. Listener Definitions

Listeners are Java classes that perform tasks on specific events sent by the application server, like application startup and shutdown or session creation and deletion. The listener

section contains just the list of listeners for an application. Web Center needs just one listener.

4.2.3.1. com.siemens.webMgr.util.ContextListener

The ContextListener initializes a Web Center application on startup and cleans up resources on application shutdown.

4.2.4. Servlet Definitions and Mappings

This section describes the MetaActionServlet definitions and mappings.

4.2.4.1. MetaActionServlet

The MetaActionServlet is the main instance that controls processing of Web Center requests.

4.2.4.1.1. Servlet Name

An arbitrary name for the MetaActionServlet. The name is used in the servlet mappings section below.

4.2.4.1.2. Servlet Parameters

- **load-on-startup** - Defines when the Web Center application is loaded by the application server:
  - Positive integer or 0 – The servlet is loaded at server startup time.
  - Negative integer – The time the servlet is loaded is up to the server (default).

We recommend that you allow the servlet to be loaded at server startup time.

4.2.4.1.3. Servlet Mappings

In order to let the application server know, which Web Center requests are to be processed by the MetaActionServlet, the servlet must be mapped to the respective request URIs. For the applications provided with Web Center, map the servlet to the Struts action extension **.do**.

4.2.4.2. BinaryReaderServlet

The BinaryReaderServlet is a DirXweb for JSP servlet that serves values of binary attributes like photos and certificates. The value is returned as binary data to the browser. The servlet can for example be addressed in the href attribute of a link tag, or the src attribute of an img tag.

The sole task of the BinaryReaderServlet itself is to send binary data to the client. The servlet includes a JSP that is responsible for providing the data to the servlet in a scoped variable. The JSP might, for example, load the data from some file or read the value of a binary attribute from the directory. Shifting the details of how the data is obtained to a JSP provides greater flexibility and keeps the servlet simple and generic.

For more details, confer to the DirXweb for JSP documentation.

4.2.4.2.1. Servlet Name

An arbitrary name for the BinaryReaderServlet. The name is used in the servlet mappings section.

4.2.4.2.2. Servlet Parameters

- **JspPath** – The location of the JSP (with name **default.jsp**) to be included by the servlet, for Web Center usually **/WEB-INF/jsp/controller/binary**.

4.2.4.2.3. Servlet Mappings

Map the BinaryReaderServlet to the URI **/binaryReader/***.

4.2.4.3. SaveFileServlet

The SaveFileServlet serves reports intended to be saved as a file on the client machine. Since reports are HTML, XML, or text files, they are usually directly displayed by the browser. To prevent this, the servlet sets specific HTTP response headers causing the browser to open a dialog box that lets the user open or save the file. The headers include the file type and an appropriate file name proposal.

4.2.4.3.1. JSP File

The name of the JSP page implementing the servlet, in this case **/WEB-INF/jsp/controller/core/saveFile.jsp.**

4.2.4.3.2. Servlet Name

An arbitrary name for the SaveFileServlet. The name is used in the servlet mappings section below.

4.2.4.3.3. Servlet Parameters

- **path** – The location generated reports are temporarily stored by Web Center for being served to the browser, relative to the application's temporary directory (below Tomcat's work folder). Usually **/reports/Reports_**.
- **encoding** – The encoding for the HTTP response header Content-Type. Default: **utf-8**.

4.2.4.3.4. Servlet Mappings

Map the SaveFileServlet to the URI **/saveFile**.

4.2.5. Session Configuration

A session is created each time a user logs in to Web Center. It stores data that must be kept between requests, like the user's login credentials or entries that the user has selected. If a user does not issue any request for a specified amount of time, the application server can destroy his session. The next time the user accesses Web Center, he is redirected to Web

Center's start page and prompted to re-login.

- **session-config/session-timeout** – The number of minutes a session must be idle before the application server can destroy it. Default in Tomcat: **30**.

The value must be carefully tuned. The more sessions that exist, the more system resources are consumed, eventually slowing down performance and causing out-of-memory errors. Too short a timeout, on the other hand, will annoy the users.

Another session parameter is used to disable session tracking via URL rewriting for security reasons:

- **session-config/tracking-mode** – The way how user sessions are tracked. Should be set to **COOKIE**.

4.2.6. Welcome File List

If a user launches a Web application via a URL containing only the application's context path but no action or JSP name, the user is redirected to one of the application's welcome files.

- **welcome-file-list/welcome-file** – The paths of one or more welcome files. Each path must be specified relative to the application's context path.

The welcome file of the two applications provided with Web Center is **index.jsp**. The JSP forwards to the login page. Thus, the URL

http://*TomcatServer:TomcatPort***/webCenter**

is resolved to

http://*TomcatServer:TomcatPort***/webCenter/index.jsp**.

4.2.7. Error Handlers

Web Center assigns specific error handlers to some HTTP status codes or Java exception. This prevents Tomcat from sending its own error pages to the client. Tomcat's error pages are usually disruptive, display very generic information and are sometimes considered to pose a security risk. Web Center, on the other hand, tries to react upon errors in a non-disruptive way and display error information in the user's language.

Each error handler definition comprises the error code or exception that triggers the handler, and the URI of the handler. A handler may be a Struts action, a JSP page or an HTML page.

4.2.8. Security Settings

Security settings are used in some cases of single sign-on, like Windows single sign-on via the SPNEGO protocol. For details, see the chapters on single sign-on in the *DirX Identity Installation Guide* and the *DirX Identity User Interfaces Guide*.

4.2.9. Specific Configuration Parameters for Web Center for Password Management

The web.xml file contains some commented context parameters of external authentication that can be overridden if necessary.

- **com.siemens.webMgr.auth.varName** holds the name of variable that keeps the authentication data. Typically, does not need to be changed.
- **com.siemens.webMgr.auth.mode** holds the selected authentication mode. See “Authentication Modes” below.
- **com.siemens.webMgr.auth.userFilter** user filter that is resolved. The %LOGIN_ATTR is resolved to the login attribute name. The %USER_ID is replaced by the value entered in the login field. Default attribute (defined in loginForm) is *cn*.
- **com.siemens.webMgr.auth.masterTsAttr** holds the name of user attribute on loginForm that contains the selected master TS. By default, the attribute is not persistent. Change the name in the web.xml file and in the loginForm configuration (located in WEB-INF/configPwd/identity/forms-config.xml) to a different LDAP attribute name that should hold the persistent value at the user. Do not forget to correctly define property description in User.xml for the new attribute based on the property description of default attribute $pwdMasterTs.
- **com.siemens.webMgr.auth.addChallengesPage** defines the forward name to be used after entering missing challenges/responses within login sequence. Configure this forward in the "/login" struts action.

In case of internal authentication, the parameters *varName* and *userLoginAttr* of ExtAuthFilter are not used.

The context parameters related to external authentication are the following:

```
<context-param>
            <param-name>com.siemens.webMgr.auth.varName</param-name>
            <param-value>com.siemens.webMgr.authUserInfo</param-
value>
      </context-param>
      <context-param>
            <param-name>com.siemens.webMgr.auth.userFilter</param-
name>
            <param-
value>(&amp;(objectclass=dxrUser)(%LOGIN_ATTR=%USER_ID))</param-
value>
      </context-param>
```

Uncomment the external authentication filter with the name **ExtAuthFilter**.

Uncomment also the section External Authentication mappings.

```
<filter-mapping>
            <filter-name>ExtAuthFilter</filter-name>
            <url-pattern>/login.do</url-pattern>
      </filter-mapping>
      <filter-mapping>
            <filter-name>ExtAuthFilter</filter-name>
            <url-pattern>/login.jsp</url-pattern>
      </filter-mapping>
```

4.3. The Password.properties File

The file **password.properties** contains a number of passwords.They are stored in Java properties format (*name=value*).

If you enter a password in clear text, the server reads it during the next start-up, encrypts it and writes it back to the file.From now on the password information is no longer readable.If you are in doubt that the right password is set or if you need to set a new password, simply replace the encrypted value with the clear text value.At the next startup, Web Center encrypts the password again.

Web Center evaluates the following names:

- **ldap -** The password for the technical user; typically the domain admin.
- **ANYONE** - The password of the anonymous user (typically: *cn=ANYONE, cn=**domain*) is needed for user self-registrations.
- **webcenterKeystore** - The password of the keystore containing Web Center's private key.The private key is required for connecting to the request workflow service in case of single sign-on only.
- **webcenterKey** - The password of Web Center's private key; defaults to the keystore password.The private key is required for connecting to the request workflow service in case of single sign-on only.
- **pin** - The PIN for the current private key for decryption of attributes.The PIN is required for displaying privileged account passwords in case encryption mode is enabled.
- **previousPin** - The PIN for the previous private key for decryption of attributes.This allows smooth transition during key exchange / upgrade.Web Center is able to handle both old encrypted values (encrypted with the previous key) and new encrypted values (encrypted with the current key).

4.4. High-Level Configuration Files

This section describes high-level configuration files associated with Web Center.

4.4.1. Server-side Configuration Files

This section describes the server-side configuration files associated with Web Center.

4.4.1.1. The webCenter.properties File

The Java properties file defines a set of parameters controlling the behavior of Web Center components.

The parameters are listed in the form *name* = *value*. Any spaces around name or value are ignored. As with any Java properties file, the file accepts only characters from the character set ISO 8859-1. For characters outside Latin-1, use their Unicode escape sequence (\uxxxx), as well as for spaces (\u0020).

The file is located in folder **WEB-INF/config**. Do not change the file since it is overwritten during update installations.

For customizations, create a new file **webCenterCustom.properties** in the **WEB-INF/config** folder, copy the properties you want to customize to the new file and change their values there. Settings in **webCenterCustom.properties** take precedence over those in **webCenter.properties**.

The files are evaluated on the server side; updates to the files require a restart of Tomcat to become effective.

The configuration parameters are made available to the Web Center application in an application-scoped variable named **webCenter**. Therefore, they can be used in JSP files or struts-config.xml files with expressions like **${webCenter.resourceFolder}** and **${webCenter['dn.delim']}**.

4.4.1.1.1. Certification Campaigns

The certification campaigns configuration file has the following parameters:

- **campaigns.sizeLimit –** The size limit for listing a user's campaigns. Default: **100**.
- **campaignSubjects.sizeLimit –** The size limit for listing a user's campaigns. Default: **100**.
- **campaignNumberOfSubjects.sizeLimit –** The size limit for reading the number of subjects of a campaign. The number is displayed in a user's certification campaign list. Default: **1000**.

4.4.1.1.2. Configuration File Names

The configuration file names must be specified relative to the application's document root folder and with a leading slash.

- **config.converters** – The name of the properties file containing additional data type converter definitions for the Apache library commons-beanutils.jar (Usually **/WEB-INF/config/converters.properties**).
- **config.defaultRenderers** – The name of the properties file containing the default renderer definitions assigned to commonly used data types, editors and properties

(Usually **/WEB-INF/config/defaultRenderer.properties**).

- **config.messages** – The names of message localization files, separated by commas. Each file contains a mapping of some low-level error messages to message keys which is a last resort to localize internal messages and display them with a more user-friendly text. (Usually **/WEB-INF/config/messages.properties**).
- **customScriptsAndStyles** – The name of a custom JSP to include custom Javascript and CSS files. The custom JSP is automatically included by Web Center pages.
- **passwordFile** – The name of the file containing the directory access password of the technical users, as well as any passwords for key- and truststore access. Specify the filename relative to the Web Center application directory. (Default **/WEB-INF/password.properties**).

4.4.1.1.3. Confirmation Message Display

Web Center no longer displays messages that confirm the successful completion of an operation. With the following switch, you can re-enable the display.

- **showMessageOnSuccess –** Whether to display messages that confirm successful completion of an operation (**false**), or not (**true**). Default: **false**.

The parameter affects messages that are labeled as success messages via the **success** flag of the **setMessage** tag.

4.4.1.1.4. CSRF Filter

Settings for the cross-site request forgery (CSRF) filter.

- **csrf.enabled –** Whether the filter is enabled (**true**) or not (**false**). Default: **true**.
- **csrf.allowPostForEntryPoints –** Whether entry points are restricted to HTTP GET requests (**false**) or also evaluated for POST requests (**true**). Default: **false**.
- **csrf.tokenName –** The HTTP parameter name of the random tokens. Default: **WT**.
- **csrf.tokenLength –** The length (number of bytes) of the random tokens generated by the class **java.security.SecureRandom**. Note that the tokens in the HTTP parameters have double the length due to encoding. Default: **20**.
- **csrf.tokenScope –** Whether to change tokens per request (**request**) or per session (**session**). Default: **request**.
- **csrf.maxRedirectURLs –** The maximum size of the list of expected URLs of HTTP redirect requests from one Struts action to another one. Default: **5**.
- **csrf.entryPoints –** The comma separated list of entry points to be exempted from the token match. The list of entry points required for all Web Center applications includes:

  ```
  /login.do,/index.jsp,/error.do,/logout.do
  ```

  Add only safe actions to the list; in particular, no actions that make any modifications in the database.

4.4.1.1.5. Development Mode

Web Center runs either in productive mode or development mode.

- **developmentMode** – Switches on development mode (**true**) or productive mode (**false**). Default: **false**.

In development mode:

- Code snippets are reloaded on each access. Changes to a snippet become effective immediately without having to restart Tomcat.
- Web Center error pages may display more detailed error information like stack traces.
- In addition to this, you can have Web Center reload message files on each request:
- **developmentMode.resetBundles –** Reload message files **text.properties**, **text_en.properties** and the like on each request (**true**) or just once on application start (**false**). Default: **false**.

  In productive mode:

- Code snippets are loaded once on first access, cached in memory and served from the cache on each subsequent access.
- Web Center error pages do not display detailed error information like stack traces.
- The message files are loaded once. Changes to the files require an application reload to take into effect.

  For performance and security reasons, you should not enable the development mode in a productive system.

4.4.1.1.6. Digital Signing

Support for digital signing of client requests via the CAPICOM API has been abandoned by Microsoft as of operating system Windows 7. The new alternative solution is based on a Java applet.

- **useSigningApplet** – Sign requests via a Java applet (**true**) or via Microsoft CAPICOM (**false**). Default: **false**.

4.4.1.1.7. Distinguished Name Representation

This set of parameters defines how distinguished names are represented in export formats of Web Center tables. For details see the chapter on DN representation in the Web Center Customization Guide.

- **dn.delim** – The delimiter between adjacent nodes. Default: **“, “** (a comma, followed by a space).
- **dn.excludeTopNodes** – The number of top nodes to be excluded from the representation. Default: **1**.
- **dn.oldStyle** – Whether to use the traditional DN representation; if true all other configuration parameters are ignored. Default: **false**.

- **dn.prefix** – A prefix for the representation. Default: **“”** (the empty string).
- **dn.reversed** – Whether to list the nodes in reversed order (from top to bottom). Default: **false**.
- **dn.types** – Whether to display name part types. Default: **false**.

4.4.1.1.8. File Upload

File upload is used to upload the content of binary attributes like photos and certificates to the server.

- **binaryData.maxContentLength** – The maximum HTTP content length for upload request (number of bytes). This is mainly the total size of the files to upload, plus some bytes to transfer the file names and the like. Default: **100000**.
- **binaryData.contentTypes** – For each binary attribute, the list of accepted HTTP content types. Default: **none**.

  The syntax is

  ```
  attributeName=contentTypeList+attributeName=contentTypeList+..
  ```

  Each contentTypeList is a comma-separated list of content types.

4.4.1.1.9. Footer Display

The parameters define which parts of the footer to display on each page:

- **footer –** Whether to display the footer section at all (**true**) or not (**false**). If not, none of the sections below are visible. Default: **true**.
- **footerRuler –** Whether to display the horizontal ruler above the footer (**true**) or not (**false**). Default: **true**.
- **footerLeft –** Whether to display the left part of the footer (**true**) or not (**false**). The left part usually shows the product version. Default: **true**.
- **footerCenter –** Whether to display the center part of the footer (**true**) or not (**false**). The center part usually shows the copyright. Default: **true**.
- **footerRight –** Whether to display the right part of the footer (**true**) or not (**false**). The right part usually shows the product suite. Default: **true**.

See the *DirX Identity Web Center Customization Guide* for samples.

4.4.1.1.10. Form Display

Summary and edit forms are displayed with a frame and a toolbar, which can be globally enabled or disabled.

- **formReadOnlyTabIndex** – The tab index for read-only form fields.
  - **0** – Include read-only fields in the tab order; default.

    - **-1** – Exclude read-only fields from the tab order.
- **formShowFrames** – Whether to display list frames (**true**) or not (**false**). Default: **true**.
- **formShowToolbars** – Whether to display list toolbars (**true**) or not (**false**). Default: **true**.
- **formShowToolbarBorders** – Whether to display a border around each tool (**true**) or not (**false**). Default: **false**.
- **formShowToolbarLabels** – Whether to display labels alongside the tool icons (**true**) or not (**false**). Default: **false**.
- **formShowToolLabelPrefix** – Whether to display generic ARIA label prefixes for tools (**default**), toolbar specific prefixes (**true**) or no prefix at all (**false**). Default: **default**.

4.4.1.1.11. Group Member Listings

When the group member list is requested, the members are fetched in chunks from the database instead of reading them one by one. This method improves performance significantly. Configuration parameters for listing group members are:

- **groupMembers.sizeLimit** – The maximum number of members to display. Default: **10000**.
- **groupMembers.accountChunkSize** – The chunk size for reading account members. Default: **1000**.
- **groupMembers.userChunkSize** – The chunk size for reading user members. Default: **250**.

4.4.1.1.12. Header Display

These parameters define which parts of the header to display on each page.

- **headerTop –** Whether to display the top part of the header (**true**) or not (**false**). Default: **true**.

  The top part displays a background picture and includes

    - The key visual.
    - The company logo or name.
    - The product name.
    - The welcome text.
    - The options.
    - Links to logout or to cancel a registration.

  All parts are optional.

- **headerKeyVisual –** Whether to display the key visual (**true**) or not (**false**). Default: **true**.
- **headerCompanyLogo –** Whether to display the company logo (**true**) or not (**false**). Default: **false**.
- **headerCompanyName –** Whether to display the company name (**true**) or not (**false**).

The name is not visible if the company logo is displayed. Default: **true**.

- **headerProductName –** Whether to display the bottom row of the header (**true**) or not (**false**). The bottom row usually shows the application's menu. Default: **true**.
- **headerWelcome –** Whether to display the welcome text (**true**) or not (**false**). Default: **true**.
- **headerOptions –** Whether to display the options (**true**) or not (**false**). If not, neither the language chooser nor the font size chooser is visible. Default: **true**.
- **headerOptionsLanguage –** Whether to display the language chooser (**true**) or not (**false**). Default: **true**.
- **headerOptionsFontSize –** Whether to display the font size chooser (**true**) or not (**false**). Default: **true**.
- **headerLinks –** Whether to display the link section (**true**) or not (**false**). The section displays links to logout or to cancel a registration. Default: **true**.
- **headerLogoutLink –** Whether to display the logout link in the bread crumb navigation (**true**) or not (**false**). Default: **false**.
- **headerBottom –** Whether to display the bottom of the header (**true**) or not (**false**). The bottom usually shows only a background picture. Default: **true**.

See the *DirX Identity Web Center Customization Guide* for samples.

4.4.1.1.13. Home Page

The home page comprises a configurable set of plug-in pages which may display property pages or entry lists. For some preconfigured lists, you can define the maximum number of entries to be displayed.

- **homeMaxNumberOfAccounts** – The maximum number of accounts to be displayed in the account list. Default: **5**.
- **homeMaxNumberOfCampaigns** – The maximum number of certification campaigns to be displayed in the campaign list. Default: **5**.
- **homeMaxNumberOfGroups** – The maximum number of groups to be displayed in the group list. Default: **5**.
- **homeMaxNumberOfPermissions** – The maximum number of permissions to be displayed in the permission list. Default: **5**.
- **homeMaxNumberOfRoles** – The maximum number of roles to be displayed in the role list. Default: **5**.
- **homeMaxNumberOfTasks** – The maximum number of tasks to be displayed in the task list. Default: **5**.

4.4.1.1.14. Input Validation

The input validation filter checks the parameters of each incoming HTTP request. If it encounters a parameter value that contains a JSP expression, it rejects the request.

- **inputValidation.excludedParameters** – The comma-separated list of the names of

parameters to be excluded from validation. The default list is **password,oldPassword,retypedPassword,newPassword**.

- **inputValidation.validateRequestParameters** – Whether input validation is enabled (**true**) or disabled (**false**). Default: **true**.

4.4.1.1.15. List Configuration

Whenever you go to a Web Center page that displays a list with a search panel, the list may be automatically pre-filled by searching for entries that match the default search criteria or the ones from the previous search. Alternatively, the list may be initially empty and only filled after you've clicked the search button.

- **initialSearch** – Defines the default list behavior. If **false**, the lists will be initially empty. Default: **false**.

Note that you can override the default behavior per Struts action displaying a list by setting a request-scoped attribute with the name **initialSearch** to **true** or **false**.

The search panel above an item list may offer searching for attributes with DN syntax, like user manager or a role owner. The searches are configured in a separate configuration file, but can be globally enabled or disabled here.

- **searchForDNAttributes** – Whether searches for DN syntax attributes are enabled (**true**) or disabled (**false**). Default: **true**.

Many lists are displayed with a frame, a toolbar and a control to select the page size. Frames and toolbars can be globally enabled or disabled. Also, the page size selector items can be defined.

- **listShowFrames** – Whether to display list frames (**true**) or not (**false**). Default: **true**.
- **listShowToolbars** – Whether to display list toolbars (**true**) or not (**false**). Default: **true**.
- **listShowToolbarBorders** – Whether to display a border around each tool (**true**) or not (**false**). Default: **false**.
- **listShowToolbarLabels** – Whether to display labels alongside the tool icons (**true**) or not (**false**). Default: **false**.
- **listPageSizeItems.<name>** – The page size selector items, a Javascript array. Each array item is either a Javascript number (a page size), or the string "all" (display all items on a single page). Each selector has a name which is referenced in the tile definitions of the item lists. The standard Web Center applications use the name "default":

  ```
  listPageSizeItems.default = [25,50,100,'all']
  ```

- **listShowToolLabelPrefix** – Whether to display generic ARIA label prefixes for tools (**default**), toolbar specific prefixes (**true**) or no prefix at all (**false**). Default: **default**.

4.4.1.1.16. Login Configuration

You can customize the login procedures of Web Center via the following parameters:

- **loginChangeSessionId** – To prevent session fixation attacks, Web Center changes the session ID after successful logins. This switch serves to enable (**true**) or disable (**false**) the feature. Default: **true**.
- **preventLoginViaGet** – This switch serves to reject (**true**) or process (**false**) attempts to login with username and password via HTTP GET requests. Default: **true**.

4.4.1.1.17. Login Cookie Configuration

The login cookie might be considered to reveal security compromising information under some circumstances. The following parameters let you customize the content and behavior of the cookie.

- **loginCookie.enabled –** Whether to enable (**true**) or disable (**false**) the login cookie. Default: **true**.
- **loginCookie.includeAttributes –** Whether the cookie should include the values of the user identification fields entered into the login form (**true**) or not (**false**). Default: **true**.
- **loginCookie.maxAge –** The maximum lifetime for the login cookie (in seconds). Specify 0 to have the browser discard the cookie immediately. If left unspecified the cookie is discarded when the browser is closed. Default: **2592000** (30 days).
- **loginCookie.secure –** Whether to send the cookie over secure HTTPS connections only (**true**) or over any connection whether secure or insecure (**false**). Default: **false**.

4.4.1.1.18. Login Form Configuration

Some parameters let you customize the login page and form.

- **loginForm.autoComplete** – Whether to enable (no value) or disable (**off**) the browser's autocompletion feature for the user identification fields in the login form. The parameter applies to each form field with renderer **secureText**. The autocompletion feature should be disabled for security reasons. Default: **off**.
- **loginForm.minChars** – The minimum number of characters (besides wildcards and spaces) to be entered as login name. If set to 0, the users are required to enter their exact login names. Default: **0**.
- **loginForm.showRegister** – Whether to show (**true**) or to hide (**false**) the self-registration section on the login page. Default: **false**.

You can also customize the search for the DirX Identity user matching the identification data entered into the login form:

- **loginForm.searchBase –** The search base; the default value is taken from context parameter **com.siemens.webMgr.ldap.baseDN** in file **web.xml** which is usually the DXI domain.
- **loginForm.searchFilter –** The search filter; the default filter is "(objectClass=dxrUser)". The filter will be extended at runtime by the identification data entered into the login

form, for example "(&(objectClass=dxrUser)(cn=taspatch nik))".

4.4.1.1.19. Login via Challenge/Response Configuration

You can customize login to Web Center via challenge/response with the following parameters:

- **challengeResponses.duplicateResponsesAllowed –** Whether users can specify identical responses for different authentication questions (**true**) or not (**false**). Default: **false**.
- **challengeResponses.minimumResponseLength –** The minimum response length. Default: **1**.
- **challengeResponses.trimOnAnswering –** Whether leading and trailing spaces are removed from responses when answering authentication questions (**true**) or not (**false**). Default: **true**.
- **challengeResponses.trimOnEditing –** Whether leading and trailing spaces are removed from questions and responses when defining authentication questions (**true**) or not (**false**). Default: **true**.
- **editableChallenges –** Whether users can define their own authentication questions (**true**) or just select from the list of predefined questions (**false**). Default: **true**.
- **loginMinChallenges** – The minimum number of questions a user is required to answer correctly if he forgets his password. Users with an insufficient number of registered challenges cannot login this way. Default: **2**.
- **minEnteredChallenges** – The minimum number of authentication questions that a user is required to enter when he registers. The default is **6**.

4.4.1.1.20. Menu Configuration

The standard Web Center layout page can display the menu horizontally in the page header, or vertically on the left side of the content area.

- **menuVertical** – Defines whether to display a vertical (**true**) or horizontal (**false**) menu. Default: **false**.
- **menu.argumentMarker** – The string that marks arguments in menu item labels. The marker is removed for empty arguments. Default: **“:\u0020”**.
- **menu.showInactiveItems** – Whether to show inactive menu (**true**) items or not (**false**). Default: **true**.
- **menu.showLabelPrefix** – Whether to display ARIA label prefixes for menus (**true**) or not (**false**). Default: **true**.

4.4.1.1.21. Navigation History

- **navigationHistory.maxItems –** The maximum number of navigation history items. Default: **15**.

4.4.1.1.22. Password Management

This section comprises configuration parameters which apply to Web Center for Password Management only. For details consult the relevant use case document.

- **passwordManagement.login.syncUserPassword** – Defines whether the user password is synchronized (**true**) after login or not (**false**). Default: **true**.
- **passwordManagement.login.maxNumCheckStatus** – The maximum number of times the status of a password synchronization is checked before giving up. Default: **10**.

4.4.1.1.23. Password Management Mode

The password management mode defines how Web Center processes password modifications requested by a user or administrator.

- **passwordMode** – The following modes are available:
  - **directory** – Passwords are immediately stored in the directory standard attribute userPassword but not in the DirX Identity attribute dxmPassword. This mode cannot be used for password synchronization to target systems (it is only possible to synchronize the passwords if the target system uses the same hash format as the DirX Identity directory). Password policies are ignored.
  - **classic** – Passwords are immediately stored in the directory standard attribute userPassword and the DirX Identity attribute dxmPassword. Additionally, a message is sent to trigger synchronization to target systems.

    If the directory or the messaging server is not present for some time this might result in inconsistencies. Thus we recommend using the "identity" mode. Password policies are ignored.

  - **identity** (default) – Passwords are not stored in DirX Identity store directly. Instead, Web Center sends a message to the event manager that performs this task later on.

The password mode may affect the visibility of menu items in Web Center's navigation bar.

4.4.1.1.24. Popup Window Scripts

For security reasons, the list of script files to be included in popup window pages is no longer directly taken from an HTTP request parameter but defined on the server instead. The HTTP request specifies only the script list identifier.

- **scriptList.tree** – The scripts to be included in tree windows.
- **scriptList.roleParams** – The scripts to be included in role parameter windows.

4.4.1.1.25. Privilege Assignments

Available Privileges Filter

Assigning privileges to business objects is not subject to approval. This means that users

always inherit privilege assignments via business objects without approval. Therefore, Web Center by default allows assigning only those privileges to business objects that are not subject to approval when directly assigned to users. This is achieved via the additional filter **(dxrNeedsApproval=false)** that applies whenever Web Center searches for privileges assignable to business objects.

- **assignPrivilegesToBO.additionalFilter** – An additional filter to be applied when searching for privileges assignable to business objects.

Request Reason

On assigning or withdrawing privileges, the requestor can enter a reason. The reason is then displayed later on to potential approvers. You can enable or disable the reason field for specific types of assignment pages.

- **requestReason.addUsers.enabled** – Display the reason field when adding one or more users to a privilege (**true**).
- **requestReason.removeUsers.enabled** – Display the reason field when removing one or more users from a privilege (**true**).
- **requestReason.assignPrivilegesToBO.enabled** – Display the reason field when assigning privileges to or withdrawing privileges from a business object (**true**).
- **requestReason.assignPrivilegesToPrivilege.enabled** – Display the reason field when assigning privileges to or withdrawing privileges from a role or permission (**true**).
- **requestReason.assignPrivilegesToUser.enabled** – Display the reason field when assigning privileges to or withdrawing privileges from a user (**true**).

4.4.1.1.26. Search Panel Configuration

Some default search panel settings. The settings may be overwritten per search panel in **tiles-defs.xml** files.

- **searchFilterConjunctions –** The conjunctions available in search panels:
- **and;or –** Both conjunctions are selectable; default conjunction is “and”.
- **or;and –** Both conjunctions are selectable; default conjunction is “or”.
- **and –** The only filter conjunction is “and”.
- **or –** The only filter conjunction is “or”.

  Default: **and;or**.

- **searchFilterCriteriaCount –** The number of search filter rows. If the number of rows is 0, a single row is displayed initially, and rows may be added or deleted via respective buttons. If the number is greater than 0, exactly that number of rows is displayed while adding and deleting rows is disabled. Default: **0**.
- **searchFilterOperands –** The operands to be selectable in the operands combo box. The operands vary with the filter attribute. The supported operands are **beginsWith**, **contains**, **endsWith**, **equals**, **greaterOrEqual**, **lessOrEqual**, and **isPresent**, and their negated counterparts, like **not.contains**. Each operand may be followed by the key of its

label, and the applicable attribute types, like **date**, **time**, **number**, **bool, list, text, noSubs, sublink** and **all**. Default: **beginsWith;contains;endsWith;equals::all; greaterOrEqual::date,time,number;lessOrEqual::date,time,number;isPresent::all**.

Similar configuration parameters apply to search panels on assignment pages.

- **searchFilterCriteriaCountForAssignments –** The number of search filter rows. Default: **1**.
- **searchFilterConjunctionsForAssignments –** The conjunctions available in search panels. Default: **and;or**.
- **searchFilterOperandsForAssignments** – The operands to be selectable in the operands combo box. Default: **beginsWith;contains;endsWith;equals::list;isPresent::list**.

4.4.1.1.27. Scheduled Change Management

Scheduled change management lets you perform operations that become effective only on some future date. Web Center displays a control to set the due date alongside page submit buttons. You can enable or disable the control per operation type.

- **dueDate.enabled** – A general switch to enable (**true**) or disable (**false**) the due date control. If disabled, the control is never displayed. If enabled, the display depends on the operation specific switches. Default: **true**.
- **dueDate.assign.enabled** – Enables (**true**) or disables (**false**) the due date control for assignment forms. Default: **true**.
- **dueDate.create.enabled** – Enables (**true**) or disables (**false**) the due date control for create forms. Default: **true**.
- **dueDate.createOrModify.enabled** – Enables (**true**) or disables (**false**) the due date control for form that may perform both modify and create operations. Default: **true**.
- **dueDate.delete.enabled** – Enables (**true**) or disables (**false**) the due date control for delete forms. Default: **true**.
- **dueDate.modify.enabled** – Enables (**true**) or disables (**false**) the due date control for modify forms. Default: **true**.

Note that the due date operation type of a form is defined by the label key of the due date control in the respective form-bean configuration or, for delete operations, in the respective menu configuration.

4.4.1.1.28. Single Sign-On

When a user logs in to Web Center with name and password, Web Center checks the user's password status attributes (reset flag, expiry date) and takes appropriate actions in case the password must be reset or is about to expire.

In case of single sign-on, the checks are disabled by default. Use this parameter to activate the checks for SSO authentications.

- **ssoCheckPwdStatus** – Whether to check password status (**true**) or not (**false**) in case of single sign-on. Default: **false**.

A single sign-on request fails if the provided single sign-on user data cannot be mapped to a user in the DirX Identity user database. In that case, Web Center falls back to the standard login page with username and password. Alternatively, you can instruct Web Center to reject the request altogether.

- **ssoFallbackToLoginPage –** Whether to display the standard login page (**true**) or to reject the request (**false**). Default: **true**.

A single sign-on component may propagate the user language to Web Center via an HTTP request parameter.

- **languageParameterName –** The name of the HTTP request parameter.

4.4.1.1.29. Size Limits

The general size limit for searching the affected users of a privilege.

- **listPrivilegeUsers.sizelimit –** The size limit. Default: **500**.

The size limit for searching the affected users of a privilege when generating reports. This limit applies only if the general size limit is 0.

- **report.sizelimit –** The size limit. Default: **1000**.

The size limit for searching roles while searching the affected users of a role.

- roles.sizelimit – The size limit. Default: **500**.

The size limit for listing children of an entry in a tree view.

- **listChildren.sizelimit –** The size limit. The default value is the general LDAP search size limit configured at the domain.

4.4.1.1.30. SoD Violations

Web Center can display SoD violations reported by an external SoD provider. The list of providers is extendable.

- **externalSODViolationProviders** – A comma-separated list of SoD providers. For each provider, the list must contain the Java class required to access the provider; note that the class must implement a specific Java interface.

4.4.1.1.31. Static Resources

This section comprises some parameters to customize static resources like icons and logos. The file and folder names must be specified relative to the application's document root folder (no leading slash).

- **resourceFavicon** – The application's favorite icon. Browsers display the icon in their address bar or along with bookmarks to the application. Windows displays the icon along with links from the file system to the application. Usually: **./resources/images/logos/DirXIdentity.ico**.

- **resourceFolder** – The resource folder name. The folder depends on the DirX identity build version in order to get new resources automatically loaded to the clients after upgrades. Usually **./resources/<build>**.
- **resourceLogo –** The file name of the company logo. Usually: ./resources/images/logos/logo.png.

Another set of parameters specifies the file encodings of language specific HTML text snippets in the folder **WEB-INF/resources/languages/***language*. By default, Web Center reads these files with the default Java file encoding.

- **resources.html.fileEncoding.***language* – The file encoding of the HTML text snippets for language *language*, for example
- resources.html.fileEncoding.de = ISO-8859-1
- resources.html.fileEncoding.en = ISO-8859-1
- resources.html.fileEncoding.ja = UTF-8

4.4.1.1.32. Struts

The Struts Validator Plug-In is disabled by default for security reasons.

- **struts.validator.enabled –** Whether the validator is enabled (**true**) or not (**false**). Default: **false**.

4.4.1.1.33. Style Sheets

The Web Center style sheets are delivered in 3 variants with different font sizes.

- **availableStyles –** The comma separated list of all available styles (small,medium,large).
- **defaultStyle** – Selects the default style sheet variant (**small**, **medium,** or **large**).

A user may select a different style sheet, which is then stored in the login cookie that is permanently stored in the browser until the cookie expires. The cookie overrides the default style on subsequent requests.

4.4.1.1.34. Utility Bar

The parameters define which components to display in the utility bar on each page.

- **utilityBar –** Whether to display the utility bar (**true**) or not (**false**). If not, no utility is visible. Default: **true**.
- **utilityNavigationHistory –** Whether to display the navigation history in the utility bar (**true**) or not (**false**). Default: **true**.
- **utilityQuickSearch –** Whether to display the quicksearch utility (**true**) or not (**false**). Default: **true**.
- **utilityAdvancedSearch –** Whether to display the advanced search link (**true**) or not (**false**). Default: **true**.

4.4.1.1.35. Workflows and Task List

A global DirX Identity flag lets you specify whether approvers can change data on approval pages. To implement this feature, Web Center must be able to distinguish approval activities from other ones:

- **tasks.approvalActivityTypes –** The comma separated list of approval activity types. Usually: approveCreate,approveModification,approveAssignment.

The next group of parameters controls how a user's task list works:

- **tasks.sizeLimit –** The maximum number of entries to be displayed in a task list. Minimum: 1. Default: 100.
- **tasksAutoOpenNext** – Whether to go back to the task list when the user has completed a task (**false**), or whether to automatically open the next task (if any) (**true**). Default: **false**.
- **tasksAutoOpenSingle** – If a user opens his task list and the list contains just a single task, open the task (**true**) or display the task list (**false**). Note that when the task is opened automatically the user is not able to perform list context menu operations (like change participant) on the task. Default: **false**.
- **tasksShowOkMessageOnCompletion** – Whether to display a message on sucessful task completion (**true**) or not (**false**). Default: **false**.

Another parameter defines whether the content of an approval activity which is part of a self-registration workflow is always editable even when the read-only flag at the approval activity is set:

- **tasks.selfRegistration.approvalContent.alwaysEditable** – The content is always editable (**true**) or is editable only if the read-only flag isn't set (**false**). Default: **false**.

Web Center takes the labels for the attributes of an enterAttributes or approveCreate activity by default from file **text.properties**, while labels configured in the activity definition are ignored. To change the default behaviour use the flag

- **tasks.useLabelsFromDefinition** – Whether to use the labels from the activity definition (**true**) or not (**false**). Default: **false**.

The next parameter affects the display of workflow lists:

- **workflowListPeriod** – Display successfully finished or failed workflows only if their end dates do not date back more than the number of days specified here. Default: **14**.
- **workflowListSizeLimit** – The client-side size limit for a workflow list; specify **0** for unlimited. Default: **250**.

The next parameter affects the display of workflow details pages:

- **workflow.details.showCancelledActivities** – Whether to show cancelled activities in the list of finished activities (**true**) or not (**false**). Default: **false**.

The following parameters serve to exclude workflow definitions from create object

workflow definition lists.

- **workflows.create.***objectType***.exclude** – Exclude items matching the conditions.
- **workflows.create.***objectType***.include** – Exclude items not matching the conditions.

Each condition defines a regular expression to be matched by the name, description,approve or path of workflow definitions: **name:***expression*, **description:***expression* or **path:***expression*. Conditions are separated by semicolons.

When assigning privileges to a user or to another privilege or when assigning a privilege to a list of users, Web Center generates a correlation ID and stores it in the context of all approval workflows triggered by the assignments. The correlation ID can then be used to identify approval workflows which originated from the same action in Web Center:

- **workflowCorrelationIdName** – The name of the workflow context attribute for the correlation ID. The default name is **correlationId**. The name **none** disables correlation IDs.

4.4.1.2. The webCenter-FileUpload.properties File

A set of parameters that control how Web Center's file upload feature operates. For details, see the related Use Case document.

4.4.1.3. The paths.properties File

The file **paths.properties** contains a couple of action and JSP lists. The lists were defined in checkSession.jsp in previous versions but have been moved to a separate configuration file for easier adaptation.

For customizations, create a new file **pathsCustom.properties** in the **WEB-INF/config** folder, copy the properties you want to customize to the new file and change their values there. Settings in **pathsCustom.properties** take precedence over those in **webCenter.properties**

4.4.1.3.1. cancel

Actions and JSPs cancelling an operation. Used to avoid annoying "Your session is invalid" messages.

4.4.1.3.2. challengeResponse

Actions and JSPs allowed while trying to login via challenge/response.

4.4.1.3.3. forcePasswordChange

Actions and JSPs allowed while user is forced to change is password.

4.4.1.3.4. initial

Initial actions and JSPs not presuming a valid session.

4.4.1.3.5. selfRegistration

Actions and JSPs allowed during user self-registration.

4.4.2. Client-side Configuration File config.js

The file defines a Javascript object with parameters to control the behavior of some user interface components. The object is composed of sub objects which comprise sets of related configuration parameters.

The file is located in folder **resources/<build>/config**.

The file is evaluated in Javascript code running in browsers; updates to the file require a browser refresh to take into effect.

Access to the configuration parameters in Javascript files is provided by the method **config.get**, that is **config.get("dn.delim")**.

4.4.2.1. Distinguished Names

The sub object with name **dn** controls how distinguished names are represented in the user interface. For details, see the chapter on DN representation in the *DirX Identity Web Center Customization Guide*.

4.4.2.1.1. Parameters

- **delim** – The delimiter between adjacent nodes. Default: **", "** (a comma, followed by a space).
- **excludeTopNodes** – The number of top nodes to be excluded from the representation. Default: **1**.
- **oldStyle** – Whether to use the traditional DN representation; if true all other configuration parameters are ignored. Default: **false**.
- **prefix** – A prefix for the representation. Default: **""** (the empty string).
- **reversed** – Whether to list the nodes in reversed order (from top to bottom). Default: **false**.
- **types** – Whether to display namepart types. Default: **false**.
- **shortFom.ellipsis** – An appendix to the representation indicating an abbreviated form. Default: **" ..."** (a space, followed by three dots).
- **shortFom.includeNodes** – The nodes to be included in the representation. Default: **"[0,1]"**.

4.4.2.2. Forms

The sub object with name **forms** defines some form behavior.

4.4.2.2.1. Parameters

- **blockInvalid** – Whether to block submission of a form if mandatory form fields are still

empty or invalid input has been entered into some form field. Default: **true**.

- **oneTime** – Whether to block additional submissions on first form submission. Default: **true**.
- **tabIndexForScrolling –** Whether multiline read-only fields (like text areas) are included in the tab order if their content is not completely visible. Default: **true**.
- **validate** – Whether to activate client-side form input validation. Default: **true**.

4.4.2.3. InvalidInput

The sub object with name **invalidInput** sets a property of the message that is displayed on detection of invalid input fields:

4.4.2.3.1. Parameters

- **maxFields** – The maximum number of input fields for which a detailed error message is displayed. This is to prevent the error message window height to exceed the screen height in which case the window would not be properly operable. Default: **5**.

4.4.2.4. RoleParams

The sub object with name **roleParams** defines some properties of role parameter windows.

4.4.2.4.1. Parameters

- **height**
- **base –** The base role parameter window height, that is the height without all the parameters (in pixel). Default: **40**.
- **mv –** The additional height for each multi-valued parameter of type other than hierarchical DN. Default: **90**.
- **mv1 –** The additional height for each multi-valued parameter of type hierarchical DN. Default: **144**.
- **sv –** The additional height for each single-valued role parameter. Default: **90**.
- **ffMin –** The minimum window height (in pixel). Note that Firefox doesn't display too small a window nicely. The parameter is ignored in Internet Explorer. Default: **0**.
- **width** – The tree window width (in pixel). Default: **820**.
- **props** – The role parameter window features, like scrollbars and resizable. See the Javascript function Window.open for details. Default: "**scrollbars=1**".

4.4.2.5. Tables

The sub object with name **table** defines some table features.

4.4.2.5.1. Parameters

- **alwaysShowInfo** – Whether to show the number of entries found as well as first and last entry index of the currently displayed page even if a table consists of a single page only. Default: **false**.

- **defPageSize** – The default page size for paged tables. The default size is usually overridden by per-table settings in a tiles-defs.xml or forms-config.xml. Default: **15**.
- **maxPageSize** – The maximum page size for paged tables. Overrides per-table sizes that exceed the maximum size. Default: **1000**.

4.4.2.6. Trees

The sub object with name **tree** defines some tree window features.

4.4.2.6.1. Parameters

- **height** – The tree window height (in pixel). Default: **400**.
- **width** – The tree window width (in pixel). Default: **600**.
- **props** – The tree window features, like scrollbars and resizable. See the Javascript function Window.open for details. Default: "**scrollbars=1**".

4.4.2.7. Asynchronous Requests

On most events, Web Center sends a synchronous HTTP request to the server, which leads to an update of the entire page. For some events, however, Web Center can be alternatively configured to send asynchronous requests that update the content area only.

The sub object with name **async** defines some settings for these asynchronous requests.

4.4.2.7.1. Async

- **displayMessage** – Whether to display a message in case a request to update the content area was blocked (**true**) or not (**false**). Default: **false**.
- **enabledForSelect** – Whether to send asynchronous (**true**) or synchronous (**false**) requests on clicking on an entry in a list. Default: **false**.
- **enabledForMenu** – Whether to send asynchronous (**true**) or synchronous (**false**) requests on selection of functions in the main menu. Default: **false**.
- **exclusive** – Whether simultaneous requests to update the content area are blocked (**true**) or admitted (**false**). Default: **false**.
- **maxSyncScriptsTime** – Scripts included in renderer code snippets are loaded asynchronously if the snippet code itself is loaded via an asynchronous request. You can define the maximum time (milliseconds) to wait for the script to be loaded. Default: **60000**.

4.4.2.8. Asynchronous Requests: Visualizing Ongoing Requests

The sub object with the name **loading** defines some parameters of asynchronous request loading. While waiting for the response to an asynchronous request Web Center displays a small animated image in the top left corner of the browser page in order to visualize the ongoing request to the user. The animated image consists of a sequence of sub images that are displayed in round-robin fashion at a fixed rate.

4.4.2.8.1. Parameters

- **enabled** – Enables or disabled the animation. Default: **true**.
- **interval** – The time (in milliseconds) after which a sub image is replaced by the next one. Default: **500**.
- **numImages** – The number of sub images; don't change unless you provide a customized animation. Default: **8**.

4.4.2.9. Context Menu

The sub object with name **ctxMenu** defines context menu settings.

4.4.2.9.1. Parameters

- **restrictSelection** – A function in the context menu of an entry list may be applicable to a subset of all entries only (for example Task list / Approve). In that case, a button displayed along with the function name in the context menu let's you restrict the current entry selection to the entries the function is applicable to. Use this flag to enable (**true**) or disable (**false**) this feature, i.e. to turn on or off button visibility. Default: **true**.
- **selectAll** – A function in the context menu of an entry list may be applicable to a subset of all entries only (for example Task list / Approve). In that case, a button displayed along with the function name in the context menu let's you select all entries the function is applicable to while deselecting all other items. Use this flag to enable (**true**) or disable (**false**) this feature, i.e. to turn on or off button visibility. Default: **true**.
- **selectRow** – Right-clicking an entry in a list opens the context menu. This flag let's you specify whether the entry gets also selected (**true**) or not (**false**). Default: **false**.
- **timeout** – If a user selects the context menu button on a table for which no context menu is defined, Web Center displays a respective hint in a small message box. The box closes automatically after the time (in milliseconds) specified here. Default: **3000**.

4.4.2.10. Key Handling

The sub object with name **keys** defines a key handling feature.

4.4.2.10.1. Parameters

disableBackSpace – Browsers use the backspace key as a shortcut for the back button, which redisplays to the previously displayed page. This may confuse or annoy users when hitting the key inadvertently while filling out a form. Therefore, Web Center disables the backspace key on form fields when appropriate. Use this parameter to enable or disable the feature. Default: **true**.

4.4.2.11. Window Features

The sub object with name **target** defines features of windows opened by Web Center.

4.4.2.11.1. Parameters

help – Defines features of the help window. See the relevant chapter in the Web Center

Customization Guide for details.

- **key –** The key for opening the help window or bringing the help window to the front.
- **props –** The list of window features.

5. User Interface Configuration

Web Center applications are based on Struts. Customizing an application basically means customizing the Struts configuration files delivered with the standard Web Center applications. Since the files are subject to changes due to new features or bug fixes, upgrading a customization from one DirX Identity version to another one is usually a non-trivial task. The list of configuration files described in this chapter comprises:

- Struts configuration struts-config.xml
- Tiles configuration tiles-defs.xml
- Menu configuration menu-defs.xml
- Forms configuration forms-config.xml and converters.properties
- Renderer configuration renderers-config.xml and defaultRenderers.properties
- Object configuration objects-config.xml
- Validator configuration

The list of configuration files used by a specific application must be defined in the application's context descriptor; see chapter "Configuration Files" for details.

5.1. Struts Configuration struts-config.xml

The struts configuration file is the "heart" of the Web application and contains all necessary data to set up and run it.It specifies the action servlet, the form beans, action mappings and a number of plug-ins for template handling, validation, rendering and others.For an introduction to Struts configuration see the Struts documentation at http://struts.apache.org.

This section covers only the Web Center-specific Struts configuration information.

5.1.1. Form Beans

Each form referenced by the name attribute of an <action> element in this file must be declared or defined in the <form-beans> section.

A form declaration is a <form-bean> sub element specifying just form name and type, but no properties.The form must then be defined in a separate forms configuration file.Externally defined forms can be referenced from within several struts configuration files.

A form definition additionally lists the form's properties in <form-property> sub elements. A form directly defined in a struts-config.xml file cannot be referenced from within other files.

Forms of type **com.siemens.webMgr.model.DynaLocaleForm** must be defined externally. We recommend just to declare each form here, and to define it externally. For more details, see the chapter on forms configuration below.

Important attributes of the <form-bean> element are:

name – The unique identifier of the form. This value is used in the “name” attribute of <action> elements to reference the form.

type – The class name. Usually **org.apache.struts.action.DynaActionForm** or **com.siemens.webMgr.model.DynaLocaleForm**,

5.1.2. Action Mappings

Action mappings are defined in <action> elements within a <action-mappings> parent. Each <action> element maps a module-relative URI to an action class that processes the request. The following attributes are important:

path - The requested URI.

type – The class name of the action. Web Center always uses the same action handler and delegates request specific tasks to the “controller JSP page”.

name – The name of the form bean to be used in this action. It references a <form-bean> element (see the section "Form Beans" in this chapter).

parameter – A general-purpose parameter used to pass extra information to the action handler and the controller JSP page. It contains a list of name-value pairs separated by ‘;’. Any leading and trailing white spaces around parameter names and values are discarded. The first and most important parameter is “jspPage”, which denotes the “controller JSP”. The action handler copies all other parameters into the request attributes and processes the JSP page. Thus, the “controller JSP” only evaluates the other parameters and their usage should be studied there. But there are some parameters used very often:

jspPage – The relative path of the “controller JSP” page.

object – The “main” object of this action, typically the object to be displayed or edited. For example, this will be the login user in self-service requests or the role going to be modified. The JSTL notation references the object by its attribute name in the session context, for example `${sessionScope['com.siemens.webMgr.selectedUser']}`.

operation – The requested operation, for example “modify”. The operation is particularly important in case the action is triggered via a context menu. In that case, Web Center executes the action only if the logged in user has the access rights to perform the operation on the main object.

defaultSearchBase – The search base for LDAP search operations, especially used for requests to list users, roles or other objects.

elements – The object type to be searched for or a list of attributes in list operations, for example users, juniorRoles, roles.

resetPoint – A reset point (**true**) is an action to which the application can safely redirect to after processing a request to change the language or the font size. A reset point action must not perform any changes in the database, and should not rely on a specific session state.

history – Whether to include the action in the navigation history or not.

historyLabel – The message key of the label for the action's navigation history entry. The default message key is **navigationHistory.<actionPath>**.

Each <action> element typically has a number of <forward> sub-elements. The action handler and in the Web Center case most often the "controller JSP" returns a forward value. The Struts framework forwards the navigation flow to the path of the matching <forward> element. There are two different path types. A path starting with a dot names a tiles definition and instructs the controller to build the HTTP response according to that definition; see the section "Tiles Definitions tiles-defs.xml" in this chapter for details. A path starting with a slash, on the other hand, identifies a Struts action and causes page execution to be continued with that action.

5.1.3. Plug-ins

External plug-in resources can extend the Struts framework. Standard ones are the Tiles framework and a validator. Web Center also includes them and adds its own extensions.

The Web Center extensions are plug-ins for form, renderer and object configurations. Each is configured separately in a <plug-in> element like the following one for the form configuration:

```
<plug-in className="com.siemens.webMgr.config.FormConfigPlugIn"/>
```

The standard plug-ins support proprietary properties. Important are the ones where the definition files are listed.

You set the Tiles definition files in the property **definitions-config** as in the following sample snippet:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
    <set-property property="moduleAware" value="true"/>
    <set-property property="definitions-parser-validate"
value="true"/>
</plug-in>
```

In order to modularize the configuration, Web Center uses a tiles-definition file per module (identity and user management, workflows, role management, and so on). To allow Tiles to support this configuration, you must set the property **moduleAware** to **true**.

The Struts validator is registered with the following configuration:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
        value="/WEB-INF/config/validator-rules.xml,
               /WEB-INF/config/validation.xml"/>
```

```
</plug-in>
The property pathna
```

The property pathnames lists the validator configuration files.The validator plug-in is required since Web Center forms subclass the Struts class DynaValidatorActionForm.

5.2. Tiles Definitions tiles-defs.xml

The Tiles framework allows assembling presentation pages from component parts.Each part ("Tile") can be reused as often as needed throughout the application.This re-use reduces the amount of markup that needs to be maintained and makes it easier to change the look and feel of a website.

The tiles are defined in configuration files **tiles-defs.xml**.The root element <tiles-definitions> contains a number of <definition> elements.Each of them describes a tile that can be inserted in a JSP page.Definitions provide an inheritance mechanism: a definition can extend another one and override some (or all) of its attributes.

5.2.1. Element <definition>

A <definition> can define a template or a "tile".If it defines a template (or form), its “path” attribute will point to a JSP page that contains the layout of the template.If it defines a tile (or sheet), it refers to a named portion of a page or to a reusable component.

Definitions set attributes for a tile as <put> tags (see the section "Element <put>" in this chapter).

The following list shows the attributes of the <definition> element used in Web Center:

name – The unique identifier for this definition. By Web Center convention, this name should start with a period (".") to avoid naming conflicts with Struts configuration.

extends – The name of a definition that is used as the ancestor of this definition. All attributes from the ancestor are available to the new definition. Any attribute inherited from the ancestor can be overloaded by providing a new value.

path - The context-relative path to the resource used as the tile to insert. This tile will be inserted and a tiles context containing appropriate attributes will be available.

5.2.2. Element <put>

The <put> element is a child of a <definition> and describes an attribute of a definition. Its most important attributes are:

name – The unique identifier of the attribute to be set or replaced in the ancestor.

value – The value associated with this attribute.

Important <put> tags are:

action – The URL of the struts action to be performed when this form is submitted (for forms).

styleClass - The name of the CSS style class to be used for this form.

5.2.3. Web Center Base Page Definition

All JSP pages used by Web Center have a common basic tiles definition that is used to construct the final HTML page by a set of page tiles. The basic layout template is stored in jsp/view/forms/layoutPage.jsp.

The following picture gives an overview of the page areas. The content of the shaded areas is defined in separate JSP files that are included via <tiles:insert> or <c:import>.

Key visual (or Key visual + Site identifier)	Company logo	Application title
	Login status and options	
	Navigation bar	
Content area		
Footer		
[Debug area]		

The layout template contains the following areas:

- **Key visual** – The picture displayed in the top left corner of the page, and a text displayed on top of the picture.

An alternative way is to split the key visual in two part, top and bottom:

- **Key visual** – The upper part of the picture displayed in the top left corner of the page.
- **Site identifier** – The bottom part of the top left picture, overlaid with a text; a tile implemented by the JSP jsp/view/tiles/siteIdentifier.jsp.
- **Company logo** – The company logo
- **Application title** – The application's title.
- **Login status and options** – The user's login status, the logout link and option combo boxes; a tile implemented by the JSP jsp/view/tiles/subHeaderLeft.jsp.
- **Navigation bar** – The application's navigation bar which is generated by the <view:menu> tag.The navigation bar may as well be placed on the left beneath the key visual.
- **Content area** – The application's main content area; the JSP code is imported from jsp/view/forms/layoutContentArea.jsp.
- **Footer** – DirX Identity version and build number, copyright, and package (Business or Professional Suite); a tile implemented by the JSP jsp/view/tiles/footer.jsp.
- **Debug area** – Debug output area; for development purposes only.

Changing the basic layout template may be a time-consuming task, since the entire body is an HTML table with partial percentage heights and widths.Removing an item may cause the page to be displayed incorrectly.Use an HTML analog and try the changes in an HTML editor before making any layout template modification.

5.3. Menu Configuration menu-defs.xml

The menu configuration defines the menus and items for the navigation bar of a Web Center application, as well as context menus for various tables displaying entry or attribute lists.

The configuration file menu-defs.xml is a regular Tiles definitions file, registered as usual as plug-in in the main struts-config.xml file.

5.3.1. Menu Selectors

Menu selectors are flags that control the visibility of menu items.For example, some menu items are visible only if the Professional Suite is installed.Each flag has a name; its value defines the conditions under which it is true.

5.3.1.1. Menu Selectors

The menu selector list is a tiles <definition> element. Each of its attributes defines a menu selector. Attribute name is the menu selector identifier, attribute value is the condition under which the selector applies. A value usually is an expression which may use variables from application scope and evaluate the installed packages via the special variable “packages”.

5.3.1.1.1. Sample

```
<definition name=”.menuSelectors”>
    <put name="bs"
            value="${packages.Business and not
packages.Professional}"/>
    <put name="ps"    value="${packages.Professional}"/>
    <put name="pwdId" value="${webCenter.passwordMode ==
'identity'}"/>
    <put name="notFunctionalUser" value="${not apply.isFunctionalUser
            [sessionScope['com.siemens.webMgr.selectedUser']]}"/>
    <put name="runningWorkflow"   value="${not apply.isFinished

[sessionScope['com.siemens.webMgr.selectedWorkflowItem']]}"/>
</definition>
```

The section defines three menu selectors. The selector with name “bs” applies if the Business Suite is installed but not the Professional Suite. The second selector applies if the

Professional Suite is installed. And the third one applies if the password mode is set to “identity” in file webCenter.properties.

The last two selectors use the **apply** object defined in JSP page **controller/utils/initSession.jsp**.The JSP defines the pseudo-methods **isFunctionalUser** and **isFinished** which are applied here to the selected user and the selected workflow item, respectively.

5.3.2. Menus

The menus to be displayed in a navigation bar are described by a single menu bar definition. A menu bar is a list of menus, each menu a list of menu items.

5.3.2.1. Menu Bar

A menu bar is a tiles <definition> element with attributes **name** and **items**.

5.3.2.1.1. name

The menu bar name is defined by the **name** attribute of the tiles definition. The name is referenced by <view:menu> tags in view JSPs.

5.3.2.1.2. items

The semicolon-separated list of menu names. The navigation bar displays the menu titles in the same order as listed here.

The special name **sep** is intended to indicate the start of a new group of related menus; the adjacent titles will be visually separated (for example by a wider gap).

Any leading and trailing white spaces around names are discarded.

5.3.2.1.3. Sample

The following menu bar with name “.defaultMenubar” displays four menus, menu access policies permitting:

```
<definition name=".defaultMenubar">
<put name="items" value=".menuSelfService;.menuDelegation;sep;
.menuUserManagement;sep;.menuTools"/>
</definition>
```

The second menu title will be visually separated from the third one by a somewhat wider gap.

5.3.2.2. Menu

A menu is a tiles <definition> element with attributes **name**, **accessPolicyKey**, **selectors**, **msgPrefix, title, titleItem** and **items**.

5.3.2.2.1. name

The menu name is defined by the **name** attribute of the tiles definition. The name is referenced by menu bar definitions.

5.3.2.2.2. accessPolicyKey

The menu key for the menu. The key is used to check whether the logged in user has the proper menu access rights. If not, the menu is not displayed in the navigation bar. The list of available keys is stored in the directory node “cn=Keys,cn=Menus,cn=Proposal Lists,cn=Configuration,cn=*domain*”.

5.3.2.2.3. selectors

A comma-separated list of predefined values controlling whether the menu is displayed or not.

- **ps** – Skip the menu if the DirX Identity Professional Suite is not installed.
- **bs** – Skip the menu if the DirX Identity Business Suite is not installed.
- **pwdId** – Skip the menu if Web Center runs with a password management mode other than identity.

Any other selector is ignored.

5.3.2.2.4. msgPrefix

The message key prefix for the menu title and the menu items.

5.3.2.2.5. title

The message key suffix for the menu title. The entire message key is the concatenation of **msgPrefix** and **title**, separated by a dot.

5.3.2.2.6. titleItem

A menu may consist of its title, associated with an action. Clicking the title will then not pop up a list of menu items but activate the action. A sample use case is a logout menu. The value of attribute **titleItem** specifies the action as a single menu item.

5.3.2.2.7. items

The semicolon-separated list of menu items. When the menu is opened, the navigation bar displays its items in the same order as listed here.

The special item **sep** is intended to indicate the start of a new subgroup of related menu items; the adjacent items will be visually separated (eg by a horizontal line).

The special item **nl** indicates the start of a new menu row. Web Center supports one or two rows for horizontal menus. The item is ignored for vertical menus.

Any white space leading or trailing an item is discarded.

5.3.2.2.8. Sample

The following menu with name ".menuDelegation" displays three menu items, menu access policies permitting:

```
<definition name=".menu">
  <put name="title" value="title"/>
</definition>
<definition name=".menuDelegation" extends=".menu">
  <put name="accessPolicyKey"    value="Delegation"/>
  <put name="selectors"      value="ps,pwdId"/>
  <put name="msgPrefix"      value="delegation"/>
  <put name="items" value="
      showRights:/getDelegationsActive.do;
      delegateRights:/getDelegationsToDelegate.do;
      showDelegatedRights:/getDelegationsGranted.do"/>
</definition>
```

The attribute **title** is inherited from the base definition. The title message key is "delegation.title".

The menu is shown only if the DirX Identity Professional Suite is installed, and if Web Center's password management mode is **identity**.

5.3.2.3. Menu Item

A menu item definition is a string including a label, an action, a session variable, selectors and an access policy operation types. The different sections are separated by colons.

Label and action are required; session variable and selectors are optional.

5.3.2.3.1. label

The message key suffix for the menu item label. The entire message key is the concatenation of the menu's **msgPrefix** and **label**, separated by a dot.

The label is also used to check whether the logged in user has the proper access rights to the menu item. Therefore, the label must match one of the items stored in the directory node "cn=Items,cn=Menus,cn=Proposal Lists,cn=Configuration,cn=*domain*". If access is denied, the item is not displayed if the menu is opened.

5.3.2.3.2. action

The Struts action URI to be requested on clicking the menu item. The placeholder ${language} is replaced with the user's actual language (like "en" or "de") at runtime.

5.3.2.3.3. session variable

The suffix of the name of a session variable. The variable suffix is automatically prefixed by “com.siemens.webMgr.” before the variable is evaluated. The menu item is skipped if the session variable is not found or if it has an empty value. Otherwise, the value is used to replace a placeholder in the item label.

This attribute can for example be used for menu items to be displayed only if an entry like a user or role is selected, or if a search filter from a previous search is available.

5.3.2.3.4. selectors

A comma-separated list of predefined values controlling whether the menu item is displayed or not.

- **ps** – Skip the menu item if the DirX Identity Professional Suite is not installed.
- **bs** – Skip the menu item if the DirX Identity Business Suite is not installed.
- **pwdId** – Skip the menu item if Web Center runs with a password management mode other than identity.
- **mod** – Skip the menu item if the logged in user is not allowed to modify the selected entry referenced by the session variable.
- **Any other selector (except confirm)** – Skip the menu item if the value of the session variable with name “com.siemens.webMgr.<selector>” is “true”. If the selector is followed by **.inactive**, the menu item is displayed but not selectable.

There are a couple of additional selectors serving a totally different purpose:

- **confirm** – Indicates that the user has to confirm selection of the menu item, for example before deleting some entries. The key for the confirmation message text is the concatenation of the label key and the suffix “.confirm”. The default selection box button is “No”.
- **confirm.yes** – Like **confirm**, but the default button is “Yes”.
- **dueDate.operation –** Indicates that the operation supports the due date control for scheduled change management. Assign one of the due date flags in file **webCenter.properties**. If the flag is enabled, the due date control will be displayed.
- **updatePage –** Update the entire HTML page on selection of the menu item, don’t just refresh the main content area via AJAX.
- **targetName –** The name of the window in which to display the response to the request resulting from clicking the menu item.
- **targetType –** The identifier of the window properties configuration. The identifier is the name of a field in object **config.params.target** in file **resources/build/config/config.js**.

5.3.2.3.5. access policy operation types

A comma-separated list of access policy operation types from enumeration Users.Operation. The menu item is skipped if the logged-in user doesn’t have the corresponding access rights on the object referenced by the above session variable.

5.3.2.3.6. Sample

The following menu displays three menu items, menu access policies permitting:

```
<definition name=".menuAccountManagement" extends=".menu">
  <put name="accessPolicyKey"    value="AccountMgt"/>
  <put name="selectors"      value="ps,pwdId"/>
  <put name="msgPrefix"      value="accounts"/>
  <put name="items" value="
      select:/getAccounts.do;
      sep;
      summary:/showAccountData.do:selectedAccount;
      delete:/deleteAccount.do:selectedAccount:
      privilegedAccount,confirm"/>
</definition>
```

A horizontal line will be drawn between the first and second item.

The message keys for the item labels are “accounts.select”, “accounts.summary” and “accounts.delete”,

The items “summary” and “delete” are displayed only if an account is selected (since only in that case the selected account’s DN is stored in the session variable “com.siemens.webMgr.selectedAccount”).

The item “delete” is displayed only if the selected account is privileged (since only in that case the session variable “com.siemens.webMgr.privilegedAccount” is set to “true”).

The deletion of the selected account must be confirmed. The key for the confirmation message is “accounts.delete.confirm”.

5.3.3. Context Menus

The context menu for one or more tables is described by a single context menu definition. A context menu definition includes a list of context menu items.

5.3.3.1. Context Menu

A context menu is a tiles <definition> element with attributes **name**, **accessPolicyKey**, **action**, **msgPrefix** and **items**.

5.3.3.1.1. name

The context menu name is defined by the **name** attribute of the tiles definition. The name is referenced from <form-property> elements defining tables within forms-config.xml files. The name must start with a dot.

5.3.3.1.2. accessPolicyKey

The menu key for the context menu. The key is used to check whether the logged in user has the proper access rights to the menu. If not, the context menu will not be displayed. The list of available keys is stored in the directory node “cn=Keys,cn=Menus,cn=Proposal Lists,cn=Configuration,cn=*domain*”.

5.3.3.1.3. action

The Struts action to be executed on clicking one of the context menu items. The default action is the context menu name without leading dot, followed by “*.do”.

For context menus on report lists, the action must have the value “.”. Do not use that value in other cases.

5.3.3.1.4. msgPrefix

The message key prefix for the menu items.

5.3.3.1.5. items

The semicolon-separated list of context menu items. When the context menu is opened, the items are displayed in the same order as listed here.

The special item **sep** is intended to indicate the start of a new subgroup of related menu items; the adjacent items will be visually separated (for example by a horizontal line).

Any white space leading or trailing an item is discarded.

5.3.3.2. Context Menu Item

A context menu item definition is a string including a label, modifiers like a confirmation message suffix and a due date flag, and the selection flags, separated by colons. The label is required, while the other parameters are optional.

contextMenuItem := label:modifiersAndSelectors:selectionFlags

5.3.3.2.1. label

A label definition consists of a message key suffix for the menu item label and an optional access right item, separated by a comma. If the access right item is missing, it is assumed to have the same value as the message key suffix.

The label message key is the concatenation of the context menu’s **msgPrefix** and the message key suffix, separated by a dot.

The access right item is used to check whether the logged in user has the proper access rights to the menu item. Therefore, it must match one of the items stored in the directory node “cn=Items,cn=Menus,cn=Proposal Lists,cn=Configuration,cn=*domain*”. If access is denied, the menu item is not displayed in the context menu.

On selection of the menu item, the label is forwarded to the respective Struts action in the

HTTP parameter named **forward**.

The special label **ms** (multiple selections) indicates the start of the second menu section, which contains the items that apply to all selected entries. The message key for the section header is the concatenation of **msgPrefix** and **ms**, separated by a dot.

The special label **list** indicates the start of the third menu section containing the items to be applied to the entire list. The message key for the section header is the concatenation of **msgPrefix** and **list**, separated by a dot.

5.3.3.2.2. modifiersAndSelectors

Comprises flags to adjust the item's functionality and selectors to control the item's visibility and selectability. The flags and selectors are separated by commas:

modifiersAndSelectors = confirm,dueDate,sessionVariable

All flags and selectors are optional. When a selector is suffixed by ".inactive", the item is displayed in the menu but not selectable. Otherwise, the item will not appear in the menu at all.

5.3.3.2.3. confirm

Indicates that the user must confirm selection of the menu item; for example before deleting some entries. The key for the confirmation message text is the concatenation of the label key and **confirm**, separated by a dot. By default, the selected button on the confirmation page is "no". To select "yes" as the default, append **.yes** to the key.

5.3.3.2.4. dueDate

Indicates that the operation supports the due date control for scheduled change management. Assign one of the due date flags in file **webCenter.properties**. If the flag is enabled, the due date control will be displayed.

5.3.3.2.5. sessionVariable

The suffix of the name of a session variable. The variable suffix is automatically prefixed by **com.siemens.webMgr.** before the variable is evaluated. The menu item is skipped if the session variable is not found or if it has an empty value. Otherwise, the value is used to replace a placeholder in the item label.

This attribute can be used, for example, for menu items to be displayed only if an entry like a user or role is selected, or if a search filter from a previous search is available.

5.3.3.2.6. selectionFlags

A menu item may be applicable only to entries that meet some specific conditions. The menu item "approve" in a user's task list, for example, is applicable only to approval tasks, and only to those tasks that do not violate SoD. The possible conditions are defined in the contextMenuFlags attribute of the table's form bean definition; each condition is assigned an identifier. The value of **selectionFlags** lists the identifiers of the conditions to be met for this menu item.

5.3.3.2.7. Sample

The following context menu with name “.contextMenuRoles” displays five menu items, menu access policies permitting:

```
<definition name=".contextMenuRoles">
    <put name="accessPolicyKey"  value="RoleMgt"/>
    <put name="msgPrefix"        value="ctx.objects"/>
    <put name="items"
       value="summary;modify;sep;runReport;

ms:delete:confirm.roles,dueDate.delete;list;export"/>
</definition>
```

The first section has three items named summary, modify and runReport. The message key for the section header is “ctx.objects.entry”. A horizontal line will be drawn between the second and third item. The items are applied to the highlighted entry only. Their label keys are “ctx.objects.summary”, “ctx.objects.modify” and “ctx.objects.runReport”, respectively.

The second section has one item named delete. The message key for the section header is “ctx.objects.ms”. The delete item is applied to all selected entries. Its label key is “ctx.objects.delete”, Its selection must be confirmed. The confirmation message key is “ctx.objects.delete.confirm.roles”. The delete confirmation form will include the due date control if the flag **dueDate.delete** is enabled in **webCenter.properties**.

The third section has one item named export. The message key for the section header is “ctx.objects.list”. The export item is applied to the entire list. Its label key is “ctx.objects.export”,

5.3.4. Form Toolbars

A form toolbar defines a list of menu items to be visible in the toolbar above a form. The toolbar definition name is referred to in the Tiles definitions displaying the corresponding forms. The toolbar references a menu definition and specifies a list of menu items from that definition. Menu item selectors and menu access policies apply to each toolbar item as in the menu itself.

5.3.4.1. Form Toolbar

A form toolbar is a tiles <definition> element with attributes **name**, **menu** and **items**.

5.3.4.1.1. name

The toolbar name is defined by the **name** attribute of the tiles definition. The name is referenced from Tiles definitions from within tiles-defs.xml files.

5.3.4.1.2. menu

The menu definition name.

5.3.4.1.3. items

The semicolon-separated list of menu items. The toolbar displays the items in the same order as listed here.

A toolbar item may define an alias name for a menu item. The alias name is used when searching for the tool's icon, label and tooltip. The syntax is "<alias>:<menu item>".

An item may optionally be followed by one or more parameters that are transparently passed as HTTP request parameters to the server when the tool is selected: "<alias>:<menu item>:<params>". As usual, separate parameters by ampersands.

Any white space leading or trailing an item is discarded.

Sample

```
<definition name=".toolbarSelf">
    <put name="menu" value=".menuSelfService"/>
    <put name="items"

value="refresh:summary:history=false;modify;changePassword "/>
</definition>
```

The toolbar displays icons for the summary, modify and change password items of the Self-Service menu. The summary item is visualized as refresh and does not lead to an additional navigation history item.

5.3.5. List Toolbars

A list toolbar defines a list of context menu items to be visible in the toolbar above a list. The toolbar definition name is referred to in the Tiles definitions displaying the corresponding lists. The toolbar references a context menu definition and specifies a list of context menu items from that definition. Menu item selectors and menu access policies apply to each toolbar item as in the menu itself.

Note that tools are applied to entire entry lists. Therefore, the only context menu items suited out of the box for toolbars are those in context menu section "list". Other context menu items usually require implementation of specific actions.

5.3.5.1. List Toolbar

A list toolbar is a tiles <definition> element with attributes **name**, **menu** and **items**.

5.3.5.1.1. name

The toolbar name is defined by the **name** attribute of the tiles definition. The name is referenced from Tiles definitions from within tiles-defs.xml files.

5.3.5.1.2. menu

The context menu definition name.

5.3.5.1.3. items

The semicolon-separated list of context menu items. The toolbar displays the items in the same order as listed here.

A toolbar item may define an alias name for a context menu item. The alias name is used when searching for the tool's icon, label and tooltip. It is also the name of the forward the respective context menu action passes the request to. The syntax is "<alias>:<context menu item>".

Any white space leading or trailing an item is discarded.

5.3.5.1.4. Sample

```
<definition name=".toolbarRoles">
    <put name="menu" value=".contextMenuRoles"/>
    <put name="items"
value="deleteList:delete;runReportOnList;export"/>
</definition>
```

The toolbar displays icons for delete, run report and export list. The item "delete" is usually not applicable to entire lists. Therefore, it is assigned the alias "deleteList". The respective context menu action "/contextMenuRoles.do" will then pass request processing to the action assigned to forward "deleteList".

5.3.6. Start Actions

Start actions define the first page displayed to a user after a successful login. You could define a fixed page like the user search page, but this is an appropriate choice only if the user is allowed to read other user's entries, and to select menu items on them.

After a successful login, therefore, Web Center forward to the Struts action **getStartPage.do**. The action handler searches for the first menu item in the start action list the user is allowed to perform, and forwards that item provided the item matches any of the action's defined forwards. If no suitable item is found, the handler forwards "default".

5.3.6.1. Start Actions

A context menu is a tiles <definition> element with attributes **name** and **items**.

5.3.6.1.1. name

The definition name must be “.startAction”.

5.3.6.1.2. items

A semicolon-separated list of menu items. Each menu item is given by the name of its menu, followed by the item’s label. Name and label are separated by a colon.

5.3.6.1.3. Sample

```
<definition name=".startActions">
  <put name="items"
      value=".menuUserManagement:select;.menuSelfService:summary”/>
</definition>
```

The corresponding definition of **getStartPage.do** (in a struts-config.xml) is

```
<action path="/getStartAction" ...>
  <forward name=".menuSelfService:summary"
      path="/showMyUserData.do" redirect="true"/>
  <forward name="default"
      path="/getUsers.do" redirect="true"/>
</action>
```

If the user is allowed access to the item “select” in menu “.menuUserManagement“, she will get the user search page (getUsers.do).Since forward “.menuUserManagement:select” is left undefined, the “default” forward applies.

Otherwise, if the user is allowed access to the item “summary” in menu “.menuSelfService“, she will see her own entry (showMyUserData.do).

Otherwise, she will get the user search page (getUsers.do).

5.4. Forms Configuration forms-config.xml

Web Center throughout uses dynamic Struts forms of class **org.apache.struts.action.DynaActionForm** or a subclass thereof.

A dynamic form must be configured in a <form-bean> element, listing its properties in <form-property> sub elements.But there is no need to provide an extra Java Bean class with getters and setters for each of its properties.

The forms configured in a forms-config.xml file can be referenced from within one or more struts-config.xml files.Forms of class **DynaActionForm** can be defined in a struts-config.xml file as well, but then they cannot be referenced from within other struts-config.xml

files.Therefore we recommend defining each form in a forms-config.xml file.Forms of the Web Center specific subclass **com.siemens.webMgr.model.DynaLocaleForm** must be defined in a forms configuration file since they support additional attributes.

The root element <forms-config> contains the child <form-beans>, which in turn contains a list of <form-bean> definitions.The latter two elements extend the XML DTD defined in Struts.

Expressions

Most attributes in a form bean configuration do not accept expressions.

Some attributes support expressions with the full range of variables scopes: page, request, session and application. These attributes are marked with “Accepts expressions”.

```
visible="${requestScope.visible}"
visible="${sessionScope['com.siemens.webMgr.accessibility'] ==
‘readWrite’}"
```

And finally, some attributes support expressions with variable scopes limited to the form itself and to session and application scope. These attributes are marked with “Accepts form expressions”.

```
readonly="${sessionScope['com.siemens.webMgr.language'] == 'de'}"
readonly="${applicationScope['com.siemens.webMgr.licenses'].metarole}
"
readonly="${modificationForm.map.employeeType != 'Internal'}"
```

Note that the third sample may be used within the definition of the form bean with name “modificationForm” only.

Making the visibility or alterability of a form property dependent on another form property may have undesired effects if the end user is allowed to modify the master property. If, for example, the employee number is visible only if employee type is “Internal” and the user changes the type to “Customer”, the employee number will no longer be visible on a possible re-display of the page even if the change has not yet been saved to the database in between.

5.4.1. Form Beans

A <form-bean> element describes a generic form, which – in Web Center – is the content of a tile.

The form is populated with form widgets. They are configured in <form-property> elements. A widget has a text label and is usually related to an LDAP or other Bean attribute. It may be editable or read-only.

A table defines the form layout with a row index running from top to bottom and a column index running from left to right. The widgets may be grouped to form groups. The row- and column indices of such a group are related to the group origin, so they start with index 0. They do not interfere with indices from outside the form group.

The widget's HTML code is inserted into the form by a "renderer". Each widget is related to a data type, which is specified as a Java class. The data types used in Web Center have a default renderer that may be overwritten by a special field renderer for each individual widget.

The Java implementation class for a form bean derives from the **org.apache.struts.action.ActionForm** class. Nearly all form beans in Web Center are realized by the class **com.siemens.webMgr.model.DynaLocaleForm**.

Form beans and their classes are referenced by <form-bean> and <action> elements in struts-config.xml.

The following attributes are important:

name – The unique identifier for this form bean. Referenced by the <action> element in the Struts configuration to specify which form bean to use with its request. Required.

buttons – A sequence of buttonType:labelKey where buttonType must be the identifier of a button renderer configured in renderers-config.xml (submit, reset, cancel etc.) and labelKey the key for the button label. The items are separated by a ";". The attribute is also used to configure whether the form includes the due date control. To enable the control, add the name of the respective due date flag in **webCenter.properties** as first element to the button list. If you enable the flag, the form will include the due date control. Accepts expressions.

buttonGap – The distance (in pixels) between the bottom-most form property row and the submit buttons; default is **10**.

A <form-bean> element can contain any number of the following sub-elements: <form-property>, <form-property-group>, <form-include>, <form-import>. The form bean may also contain a single <form-body> element.

A <form-bean> element may also consist of an only sub-element of type <form-property-list>.

5.4.2. Form Body

The <form-body> element is intended to specify the location of the assignment panel (including the search panel) in an assignment form. Additional form properties may be placed above the assignment panel, or between assignment and button bar, or even to the left or right of the assignment panel.

The assignment properties of assignment forms are usually populated in file **assignObjects.jsp**. Other form properties are simply ignored. If you set the parameter "populateAdditionalFormFields" of the assignment action (in **struts-config.xml**) to "true", however, the JSP will also fill all other form properties with values from the current object.

5.4.2.1. Sample

The sample form displays surname and given name in the same line above the assignment panel, and manager between the panel and the button bar:

```
<form-bean name="assignRolesForm" buttons="...">
  <form-property name="sn" y="0" …/>
  <form-property name="givenName" x="2" …/>
  <form-body y="+1" spanX="4">
    <form-property name="parameters" ...>
    <form-property name="availableRoles" ...>
    </form-property>
    <form-property name="assignedRoles" ...>
    </form-property>
    <form-property name="selectedAvailable" .../>
    <form-property name="selectedAssigned" .../>
  </form-body>
  <form-property name="manager" y="+1" spanX="4" …/>
</form-bean>
```

Make sure that you set the attributes x, y and spanX correctly.

The assignment action sets paremeter “populateAdditionalFormFields”:

```
<action path="/assignRoles"
  type="com.siemens.webMgr.controller.action.JSPAction"
  name="assignRolesForm"
  parameter="jspPage:/WEB-INF/jsp/controller/core/assignObjects.jsp;

object:${sessionScope['com.siemens.webMgr.selectedUser']};
             elements:roles;
             populateAdditionalFormFields:true;
             ...">
</action>
```

5.4.3. Form Properties

The <form-property> element describes a form field.

This element is only used when the "type" attribute of the enclosing "form-bean" element is **org.apache.struts.action.DynaActionForm** or a subclass thereof.

If the field is itself a table, insert a child element <data-property> for each table column (see

the section "Data Property" in this chapter).

The important attributes compliant with the Struts schema are:

name - The name of the form field described by this element, most often the name of the LDAP attribute. Required. You can make use of placeholder names such as “$displayName”, which are defined in the object description of the corresponding object.

type – The fully-qualified Java class name of the field underlying this property, optionally followed by "[]" to indicate that the field is indexed. Required.

The following attributes are Web Center specific and extend the Struts configuration DTD:

checks – For form properties representing a table only: validations for editable table cells.

confirmation - The condition identifier and the message key of a confirmation message to be displayed when the form is submitted.

contextMenu - The tiles definition name of the context menu for a table, optionally followed by one or both of the modifiers “ms” and “list” to enable the multiselection and/or list section of the context menu. For form fields displaying object sets only.

contextMenuFlags – A list of conditions to be met by entries of a list. A condition defines one or more regular expressions to be matched by attribute values of an entry. Each condition has an identifier (a digit or a letter). The identifier is then assigned to menu items in the context menu definition. For regular expression syntax, confer to the documentation of Java class **java.util.regex.Pattern**.

dynaClass - The dyna class name for this property. This attribute is only necessary for form fields that define object sets, but not for usual attribute fields. See also the section "Data Property" in this chapter. Unless you write your own dyna class we recommend to leave this property unspecified since the default value (“com.siemens.webMgr.model.DirectoryEntryClass”) will do.

enabled – A boolean flag to enable or disable the property. The default value is "true" (enabled).

fieldRenderer – The reference to a renderer definition to be used for displaying this property; that is, for generating the HTML code of the widget. The renderer configuration file renderers-config.xml must contain an entry with this identifier. See the section "Renderers Configuration renderers-config.xml" in this chapter for details.

height – The height of the element in pixel. When defining a height take different screen resolutions into account.

hideIfEmpty – Indicates that the element is visible only if its value is not empty.

hideIfMatch – A regular expression. The form field is not visible if its value matches the expression (see documentation of Java class **java.util.regex.Pattern**).

hideOldIfEmpty – Supported by modification approval form only: Hide element for old value if old value is empty.

initialSortColumn – For form properties representing a table only: The name or index of the initial sort column.

label – The message key for the text to be displayed as form property label. The default value is “ldap.attribute.<form property name>”.

labelRenderer - The identifier string for the label renderer to be used. This attribute must be a valid entry in the renderer configuration file renderers-config.xml.

mandatory – A boolean flag to indicate whether or not the property is mandatory. The default value is "false" (not mandatory). Accepts expressions.

maxSize – The maximum number of elements to be displayed in a table. By default, all items returned by the underlying (possibly time and size limited) directory search operation are displayed. Some search operations, however, like the one for group members, are not limited.

messagePrefix – A message key prefix for form property value localization.

namingColumns – For form properties representing a table only: The comma-separated list of the names or indexes of the columns displaying the naming attributes; used for row identification in input validation error messages.

oneTimeEvents – For form properties representing a table only: events that may be triggered only once.

openAction – The Struts action to be invoked when clicking the open button of a table entry. The action may be optionally followed by a menu key and item saying that the open button should be only displayed to users that have the corresponding menu access right. Sample: /openRole.do:RoleMgt.summary. The property values accept expressions.

readonly – A Boolean flag to indicate whether the property may be edited (readonly = false). The default value is false. Accepts form expressions.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the field renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the form property. Sample: rendererProperties="targetId:accountsForm;tooltip:accounts.tooltip". The property values accept expressions.

secondarySortColumns – For form properties representing a list: The comma-separated list of column indexes or names defining the secondary sort criteria. The primary sort criterion is the selected column.

sets – A comma-separated list of set ids this property is part of. Used only for the attribute modification approval form.

size – The page size for tables. May be overridden by a corresponding Tiles definition.

sortable – For form properties representing a list: Whether the list is sortable (true) or not (false). Default: true.

spanX – The width or horizontal size of the element in number of grid cells. It is used to place the element in a virtual table grid. This attribute defines the number of table cells the element will cover in horizontal direction. The default number is 2.

spanY – The height or vertical size of the element in number of grid cells. It is used to place the element in a virtual table grid. This attribute defines the number of table cells the element will cover in vertical direction. Note that label columns are counted.

summary – The message key for the summary attribute of a table.

tooltip – The message key of a tool tip in **text.properties**. This form property only works if the associated renderer supports it, which is currently the case only for some checkbox renderers.

transient – Indicates that the form property is an artificial one not intended to be stored in the directory when saving an entry. Used for properties accepting the indexes of selected table items only.

use – The name of the form property accepting the indexes of selected items in a table.

visible – A boolean flag that indicates whether this property is to be displayed. It is used to hide properties depending on the installed license. Accepts expressions.

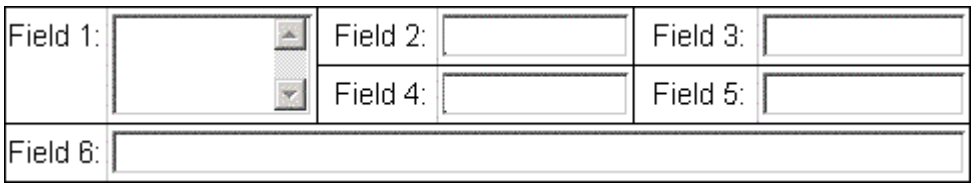
width – The width of the element in average char width (approx. 10) or pixel (depends on element type). We recommend that you specify the width of an element as a percentage, since fixed values can lead to undesirable results in different screen resolutions.

x – The relative position of the element in a row. The field is displayed to the left of all other fields with a higher x-value in the same row.

y – The vertical position of the element in coordinate points. It is used to place the element in a virtual table grid. This attribute defines the index of the table row where the upper left corner of the element is located. If omitted, the system appends this field to the same row as the previous one. A value of “+1” places the field in the next row.

Here is a sample for locating fields in a grid and setting field dimensions:

Spanned fields



Field1 spans over two rows (spanY="2"), Field6 over three columns (spanX="6", label columns are counted!). The field positions and spans should be set as follows (provided the fields are listed as sub elements of the form in ascending order from Field1 to Field6):

Field	y	x	spanY	spanX
Field1	0		2	

Field	y	x	spanY	spanX
Field2				
Field3				
Field4	+1			
Field5				
Field6	+1			6

5.4.4. Form Property Groups

Form properties can be aggregated to groups allowing a separate layout by placing them inside an element <form-property-group>. Note that fields within a property group are not aligned with fields in other property groups or outside any property group.

The <form-property-group> supports the following attributes:

ariaLabelPrefix – Whether to use the group label as ARIA label prefix for the form properties in the group.

groupRenderer – The reference to a renderer definition to be used for displaying the group of properties. The renderer configuration file “renderers-config.xml” must contain an entry with this identifier. See the section "Renderers Configuration renderers-config.xml" in this chapter for details.

hideIfEmpty – Indicates that the group is visible only if at least one of its properties has a non-empty value.

label – The message key for the text to be displayed as form property group label. The default value is “group.label.<form group name>”.

level – The group’s label level; highest and default level is 0. Only used for the attribute modification approval form.

name – The group name. The name must be unique within the form.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the group renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the form group. The property values accept expressions.

script – The name of a script file to be included within the HTTP response.

spanX – The width or horizontal size of the element in number of grid cells. It is used to place the group in a virtual table grid. This attribute defines the number of table cells the element will cover in horizontal direction. The default number is 2.

spanY – The height or vertical size of the element in number of grid cells. It is used to place the group in a virtual table grid. This attribute defines the number of table cells the element will cover in vertical direction. Note that label columns are counted.

visible – A boolean flag indicating whether this group is to be displayed. It is used to show or hide form groups depending on certain conditions. Accepts expressions.

x – The relative position of the element in a row. The group is displayed to the left of all other fields with a higher x-value in the same row.

y – The vertical position of the element in coordinate points. It is used to place the group in a virtual table grid. This attribute defines the index of the table row where the upper left corner of the element is located. If omitted, the system appends this field to the same row as the previous one. A value of “+1” places the group in the next row.

5.4.5. Form Property Lists

A form property list describes a form which displays a special kind of list like for example on a plug-in of the home page. It accepts up to three list properties as sub items.

<form-property-list> supports the following attributes:

fieldRenderer – The reference to a renderer definition to be used for displaying the list. The renderer configuration file “renderers-config.xml” must contain an entry with this identifier. See the section "Renderers Configuration renderers-config.xml" in this chapter for details.

icon – The icon to be displayed alongside each list entry; a file name relative to folder resources/images.

iconTitle – The message key for the icon tooltip.

name – The list name. The name must be unique within the form.

openAction – The Struts action to be invoked when clicking on a list item. The action may be optionally followed by a menu key and item saying that the action is only enabled for users that have the corresponding menu access right. Sample: /openRole.do:RoleMgt.summary.

openDescription – The message key for a description of the open action.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the list renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the property list. The property values accept expressions.

size – The number of elements to be displayed in the list. The preferred way is to specify the list size as a parameter of the corresponding Struts action.

sortIndexes – A comma-separated list of column indexes defining the sort order. Default is “0,1,2”.

5.4.6. Form Imports

A form may include another one. The fields and groups of the imported form are inserted

immediately after the form field, group or import preceding the import statement. The imported form will start on a new line but is not automatically terminated with a new line. The included form may itself import another one.

The <form-import> element supports the following attributes:

name – The name of the form bean to be included.

The included form must be defined before it is imported.

5.4.7. Form Includes

Form includes are used to include a form into another one. It is used only for the attribute modification approval form.

The <form-include> element supports the following attributes:

name – The name of the form to be included.

objectTypes – A comma-separated list of object types the element applies to.

workflowPaths – A comma-separated list of regular expressions matching the workflow paths the element applies to.

For a detailed description see the chapter on attribute modification approval in the *DirX Identity Web Center Customization Guide*.

5.4.8. Data Property

The element <data-property> is a sub element of the <form-property> of an object set to be displayed in a table. A <data-property> describes a single table column.

The following attributes are defined:

align – The HTML column alignment, one of **left**, **right**, **center** or **justify**.

cellRenderer - The renderer to be used for displaying the cell content of this column. This attribute must be a valid entry in the renderer configuration file renderers-config.xml. The attribute accepts expressions.

headerRenderer - The renderer to be used for displaying the content of the header cell for this column. This attribute must be a valid entry in the renderer configuration file renderers-config.xml.

label – The message key for the table column label. The default value is "ldap.attribute.<data property name>".

mandatory – A Boolean flag for editable data properties indicating whether their values may be empty (**false**) or not (**true**). The default value is **false**. The attribute accepts expressions.

messagePrefix – A prefix for message keys related to this data property.

name - The name of the data property described by this element, most often the LDAP name of an attribute of the objects to be displayed. Required. You can make use of placeholder names such as “$displayName”, which are defined in the object description of the corresponding object.

readonly – A Boolean flag to indicate whether the property may be edited (readonly = false). The default value is true.

readonlyRow – A comma-separated list of cell values that cause the entire table row containg the cell to be read-only.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the field renderer. The properties can be referenced in HTML or Javascript code snippets. Property name and value must be separated by a colon. This is a convenient way to define custom renderer properties whose values vary with the form property. Sample: rendererProperties="targetId:accountsForm;tooltip:accounts.tooltip". The property values accept expressions.

sortable – Whether the column is sortable (true) or not (false). Default: true for read-only columns, false for editable columns.

sortIndexes - A list of column numbers representing the order of sorting when this column is selected. This attribute must be written as "columnNumber, columnNumber, ...". The indexing of columns starts with 0.

transient – Indicates that the data property is an artificial one not intended to be stored in the directory when saving an entry.

type – The fully-qualified Java class name of the field underlying this property, optionally followed by "[]" to indicate that the field is indexed. Required.

visible – A boolean flag indicating whether to display this column. Accepts expressions.

whitespace – The CSS property value that specifies how to handle white space in the values displayed in this column. Sample values are **normal**, **pre** and **nowrap**.

width – The width of the column in average char width (approx. 10) or pixel (depends on element type). We recommend that you specify the width of an element as a percentage, since fixed values can lead to undesirable results in different screen resolutions.

5.4.9. List Properties

The element <list-property> is a sub-element of <form-property-list>. It describes a column in a property list.

<list-property> supports the following attributes:

cellRenderer – The renderer to be used for displaying the cell content of this column. This attribute must be a valid entry in the renderer configuration file renderers-config.xml.

label – The message key for the list column label. If no label is specified, the column is displayed without header.

messagePrefix – A prefix for message keys related to this data property.

name – The name of the form property described by this element, most often the LDAP name of an attribute of the objects to be displayed.Required.You can make use of placeholder names such as “$displayName”, which are defined in the object description of the corresponding object.

rendererProperties – A semicolon separated list of name/value pairs defining properties transparently passed to the list property renderer.The properties can be referenced in HTML or Javascript code snippets.Property name and value must be separated by a colon.This is a convenient way to define custom renderer properties whose values vary with the list property.The property values accept expressions.

type – The fully-qualified Java class name of the field underlying this property, optionally followed by "[]" to indicate that the field is indexed.Required.

whiteSpace – The CSS property value that specifies how to handle white space in the values displayed in this column.Sample values are **normal**, **pre** and **nowrap**.

width – The width of the column.We recommend that you specify the width as a percentage of the total list width, since fixed values can lead to undesirable results in different screen resolutions.

5.5. Converter Definitions converters.properties

During HTTP request processing, Struts attempts to set the properties of the form associated with the request from the HTTP request parameters.Values of request parameters, however, are always of type String.If a form property is of a different type, Struts needs to convert the String value to the respective target type.Struts uses a list of default mapping rules, based on which the conversion is accomplished for most target types.For other types, specific converters can be implemented and plugged in into Struts.

The converter definitions file contains the definition of a special converter to map HTTP request parameter values to a **java.util.Map**.The converter is registered with Struts on application start-up.

The definition maps the property type java.util.Map to the Java class implementing the conversion.Two additional arguments define how map elements are delimited in the String representation, and how key and value of each element are separated.

Sample:

java.util.Map=com.siemens.webMgr.util.MapConverter;!NI!;!NV!

The implementing Java class is **com.siemens.webMgr.util.MapConverter**. Map items are separated by “!NI!”, while keys are separated from values by “!NV!”.

For example, the request parameter value

> “My birthday!NV!111!NI!My father’s first name!NV!111!”

is converted to the Map

> {My birthday=111, My father’s first name=111}

The file is not really intended to be customized since each listed converter must implement the interface **com.siemens.webMgr.util.ParameterizedConverter.**

5.6. Renderers Configuration

Any user interface control in any form appearing in a Web Center JSP page must be converted to HTML and/or Javascript code before it can be inserted into the resulting HTML page which is then sent to the client browser.These converters are called renderers and can be configured using the file **renderers-config.xml** in the **WEB-INF/config** directory.

A renderer is associated with a Java data type.Renderers are already pre-configured for the most typical data types in
WEB-INF/config/defaultRenderer.properties.
These definitions apply, when a form property does not explicitly reference a renderer, as well as for dynamically generated request workflow forms without a form bean configuration.

Adding a new renderer (which should not be necessary in most cases) mainly requires an appropriate code snippet, which is either simply a piece of HTML code or JavaScript code.In some special cases (where the **BasicRenderer** class cannot be used), some Java programming is also necessary (as is done for the search panel or the table).

5.6.1. Renderers Configuration renderers-config.xml

The renderer configuration defines all modules producing the final HTML code for a particular page element. They are referenced from various form bean configuration elements.

The root element <renderers> contains a <renderer> child element per renderer.

5.6.1.1. Element <renderer>

The <renderer> element describes a single renderer for a particular data type. Its child elements <renderer-property> define renderer specific properties as name-value pairs.

The following attributes are defined for the <renderer> element:

id - The identifier of the renderer. This value is the reference name used in the forms configuration. Required.

extends - The id of a renderer this renderer is derived from. A renderer inherits all its attributes and properties from its superior renderer but may override one or more of them.

type - The fully-qualified Java class name of the attribute to be rendered.

className - The fully-qualified name of the renderer class. Required.

defURL - The name of the code file for the default renderer. By convention, the code accessed here constructs a renderer for read-only operations. Required.

altURL - The name of the code file for the alternate renderer. By convention, the code accessed here constructs a renderer for read-write operations.

jsURL - The name of the JavaScript code file for the table cell renderer. By convention, this code is used to render the field in a table cell. It should contain just a single line with the short name and the arguments of the Javascript function rendering the cell.

roJsURL - The name of the JavaScript code file for the table cell renderer in case the cell is read-only. By convention, this code is used to render the field in a table cell. It should contain just a single line with the short name and arguments of the Javascript function rendering the cell.

Note that code file names must be specified relative to the application's document root folder, and must start with a slash.

5.6.1.2. Element <renderer-property>

The element <renderer-property> describes renderer-specific attributes as name-value pairs. These additional attributes can be used during rendering for various purposes. Usually, such a value definition is made together with the definition of a respective placeholder ${<attribute name>} in the code snippet. During rendering the placeholder is replaced with the value defined here.

The defined attributes are:

name - The name of the property.

value - The value of the property. Note that these values will be localized. It is thus possible to use resource keys here. It is also legal to use Java-Standard-Tag-Library expressions as property values.

5.6.1.3. Sample Renderers

Important renderers are:

- **text** - Creates a text entry field in edit mode to be used to enter text content, for example. a description or a name.
- **combobox** - Creates a combo box to select values from a tag list. Note that the tag list must be in the DirX Identity object description of the attribute.
- **objectSearch** - Creates a text entry field and an action button. When the button is clicked, an action with the attribute name is called. This renderer can be used to implement a search panel to select a DN of an assigned object. The referenced object's "cn" is displayed in the text field.
- **userSearch** - Displays the attributes "sn" and "givenname" of the selected user. This renderer is very much like the objectSearch, but specialized for users.

The type String[] is often used for multi-value attributes. Some important renderers for that type are:

- **stringList** - Creates a list entry box. Each value is entered as a free string.
- **optionList** - Creates a list entry box. Each value is entered via a combo box. Note that the list of values has to be defined in the appropriate object description.
- **userLinks** - Creates a list entry box. Each value is entered via a userSearch widget.

5.6.2. Default Renderers defaultRenderer.properties

The file “defaultRenderer.properties” associates default renderers for commonly used object form property types, names, and editors.

The first file section, **Data types**, maps form property types to renderer identifiers. Form property types are class names assigned to attribute **type** of <form-property> elements in forms configuration files.

The second file section, **Editors**, maps editor class names to renderer identifiers. Editors are assigned to object properties in object descriptions.

The third file section, **Properties**, maps form property names to renderer identifiers. Form property names are class names assigned to attribute **name** of <form-property> elements in forms configuration files. If a property name is followed by an object type specification (<property name>@<object-type>, e.g. **manager@ctxlocation**), the assigned default renderer applies only to that object type. This way, you can assign different default renderers to the same property of different object types.

At runtime, the actual renderer for a form property is determined in the following order:

Form property attribute **fieldRenderer**

Form property name (section **Properties**)

Form property type (section **Data types**)

Forms used when performing a request workflow activity are not configured in forms configuration files but in request workflow definitions in the DirX Identity provisioning store. Types and field renderers for properties of these forms are looked up from property attributes type and editor in the corresponding object descriptions.

Note that the default renderer mappings are case sensitive. Class names, form property types and names, and renderer identifiers must exactly match their counterparts.

5.6.3. Renderer Code Snippets

Any renderer implementation uses code snippets to render user interface elements. These snippets are located in the **WEB-INF/snippets** directory.

A code snippet is either a piece of HTML code (for example, the simple text field) or a JavaScript (for example, all table cell renderers) or both (for example, a calendar or table). A

code snippet uses placeholders that the renderer class replaces with runtime values, thus finalizing the code before it is streamed to the client. The big advantage of these renderers is that the developer of the customized application can modify them to meet the customer's requirements.

5.6.3.1. Placeholder Value Escaping

Values substituted for placeholders may contain special characters that must be properly escaped in order not to break the generated HTML page or Javascript code. Incorrect escaping also makes a page prone to script code injection attacks by hackers when form input values or request parameters are inserted as part of the response page. The escape algorithm to apply depends on the specific location of the placeholder in the snippet.

Web Center provides a couple of escape methods. By appending a suffix to a placeholder you define which escape method to apply to the substituted value.

The supported suffixes are

- **.h –** Apply HTML escaping.
- **.jsArr –** Trim the value and check if it starts with a left and ends with a right square bracket. If not, replace the placeholder with an empty Javascript array "[]" (without quotes).
- **.jsObj –** Trim the value and check if it is enclosed in curly braces. If so, replace the placeholder with the value, otherwise with an empty Javascript object "{}" (without quotes).
- **.jsBool –** Trim the value and check if it is empty, equals "0" or "false". If so insert "false", otherwise "true" (without quotes).
- **.jsInt –** Trim the value and check if it is a sequence of digits, optionally preceeded by a "+" or "-". If so, replace the placeholder with the number, otherwise with "0" (without quotes).
- **.jss –** Apply Javascript escaping to substitutions for placeholders in Javascript strings enclosed in single quotes.
- **.jsd –** Apply Javascript escaping to substitutions for placeholders in Javascript strings enclosed in double quotes.
- **.jssd –** Apply Javascript escaping to substitutions for placeholders in Javascript strings enclosed in inner single quotes and outer double quotes.
- **.id –** Generate a valid HTML id. An HTLM id may contain only ASCII letters, digits and underscores ([a-zA-Z_0-9]). The escaper first replaces each invalid character with an underscore, and then strips leading underscores.
- **.ids –** Generate a comma-separated list of valid HTML ids
- **.subId –** Like ".id" but without stripping leading underscores.
- **.subIds –** Like ".ids" but without stripping leading underscores.

If a placeholder represents a URL, the substituted value must be properly URL encoded before applying any other escape method:

- **.url.h –** First URL encode the value, and then apply **.h** escaping.
- **.url.jss –** First URL encode the value, and then apply **.jss** escaping.
- **.url.jsd –** First URL encode the value, and then apply **.jsd** escaping.
- **.url.jssd –** First URL encode the value, and then apply **.jssd** escaping.

Javascript snippets for table cells apply the **.jss** escaping by default but support also the methods **.jsArr**, **.jsObj**, **jsBool**, **.jsInt**, **.id**, **.ids**, **.subId** and **.subIds**.

5.6.3.2. Including Script Files

HTML snippets may include Javascript or VBScript files. The way how to include the scripts depends on the type of the HTTP request. Requests that return a complete HTML page use the HTML <script> tag, for example:

```
<script
    src="${webCenter.resourceFolder}/scripts/wwPropertyList.js">
</script>
<script>WWPropList.initialize(…)</script>
```

The object WWPropList is defined in script file **wwPropertyList.js**. Since the browser executes the second script only after having loaded the file, this works fine.

However, the sample does not work for asynchronous requests that update only parts of an HTML page. The browser defers loading the script file, so that executing the second script fails since WWPropList is not yet defined. The sample works only if the file is already in the browser cache.

Web Center provides a Javascript function for including script files in asynchronous request:

```
addScript (src,type,sync,cond)
```

The first argument is the script file URL.

The second argument is the script type like “text/javascript” or “text/VBscript”; it defaults to “text/javascript”.

The third argument is the name of an object, variable or function defined in the script file. It is used to check whether the file must be loaded, and if so, to check when loading is finished. Execution of other scripts will be deferred until all script files are available.

The optional fourth argument, a boolean, indicates that the script file is required only under a certain condition. The value IS_MSIE, for example, causes the script to be included only if the browser is Internet Explorer.

The code of the above sample for asynchronous requests is

```
<script>addScript(
    "${webCenter.resourceFolder}/scripts/wwPropertyList.js",
    "","WWPropList")</script>
<script>WWPropList.initialize(…)</script>
```

Since an HTML snippet may be used in both types of HTTP requests Web Center provides a separate tag for including script files in renderer snippets. The tag is dynamically replaced with the proper code for each request.

```
<includeScript src="…" type="…" sync="…" cond="…"/>
```

The tag attributes src, type, sync and cond correspond to the arguments of function addScript. Attribute src accepts expressions, while the other ones do not.

Each includeScript tag must be on a single line and surrounded by white spaces only.

The sample code now looks like this:

```
<includeScript
    src="${webCenter.resourceFolder}/scripts/wwPropertyList.js"
    sync="WWPropList"/>
<script>WWPropList.initialize(…)</script>
```

5.6.3.2.1. HTML Snippet Samples

```
<input type="text" id="${id.id}" name="${name.h}"
       class="roTextField" style="${style.h}" value="${value.h}"/>

alertError("${value.jsd}","${confirmTo.jsd}","${confirmButtons.jsd}")
;

<input type="submit" name="${name.h}" value="${value.h}"
       onclick="return submitter.submit(

'${checkRequired.jssd}'==='false','${confirmMessage.jssd}')"/>

<table id="${name.id}" width="100%" cellpadding="0" cellspacing="0"
       class="tableBorder">

<iframe frameborder="false" width="${width.h}" height="${height.h}"
        class="${className.h}" src="${srcURL.url.h}"></iframe>
```

```
Tree.open(${data},"${params.jsd}",
    opener.document.getElementById("${targetField.jsd}"),false,false,
                                    "${loadChildrenAction.url.jsd}");

<button onclick="return postRequest('${saveURL.url.jssd}')"
                 class="labeledButton">${buttonLabel.h}</button>
```

5.6.3.2.2. Javascript Snippet Samples

```
CA("${value}",${tableInputReadOnly.jsBool},"${tableInputStyleClass}")
DNR("${value}",${includeNodes.jsArr},${link.jsBool},${reversed.jsBool
},
     "${delim}")
```

5.7. Object Configuration objects-config.xml

The object configuration mainly translates object and attribute definitions from the DirX Identity service layer to the Web Center application and back.The reason for this is somewhat technical: while many of the DirX Identity virtual object attributes contain a "." in their names, this cannot be used by the Web Center application, since it works together with the Commons bean utilities which would try to resolve "."-notations as object references.To circumvent this, we transform attribute names in the following manner:

The first term of an attribute name left to the "." character is the object set, the term on the right the relation.The objects appearing are defined as <object> elements, the relations as <relation> elements.In the Web Center application we use the name of a <relation> definition as attribute name prefix and append the name of an <object> definition with a capitalized first character.This leads to a mapping

<relation name><object name> => object.relation

Example:

Object definition: `<object id="role" …/>`

Relation definition: `<relation name="available" value=".toAssign"/>`

Web Center attribute: `availableRoles` (with the "r" of "roles" capitalized)

DirX Identity service layer attribute: `roles.toAssign`

The root element of an object configuration is <object-configuration>. It contains 2 child elements:

<**objects**> - Contains a list of <**object**> elements for each object.

<**relations**> - Contains a list of <**relation**> elements for each object relation.

5.7.1. Element <object>

The element <object> describes a single DirX Identity service layer object.

The following attributes are defined:

id – The identifier for the object. This value is the name used in attribute names. Required.

nodeClass – The fully-qualified Java class name of the service layer object. Required.

dirClass – The type, i.e. the object description name, for this object. The list of valid object types can be viewed in the DirX Identity Manager when editing access policies.

masterClass – The fully-qualified Java class name of the object's master node, which owns the object. This attribute is used especially for assignments (where the user entry class having the assignment is the master class). The attribute accepts several master classes, separated by commas.

def – Object elements may share the same combination of **nodeClass** and **masterClass**. In each a case, exactly one of the elements must be defined as default element by setting its **def** attribute to "true".

set – The set name for this object. The default set name is obtained by appending an "s" to the object identifier. There are a couple of predefined set names, like "groups", "roles" and "activities", and a general purpose set name "GenericNodes".

5.7.2. Element <relation>

The element <relation> describes the attribute mapping for a particular object relation.It requires the following attributes:

name – The name of the relation.

value – The value of the relation.Note that it must start with a '.'.

5.7.3. Search Filter Configuration

The Web Center search panel supports searching for attributes with DN syntax, like user manager or a role owner.For example, you may search for all groups whose owner has a surname starting with "ab".Web Center first searches for possible owners, that is users whose surname starts with "ab".Then it searches for all groups whose owner matches one of the candidates found.Be aware that the final search result may be somewhat surprising since the first search may hit a size or time limit.

You can define some individual and some general options for the first search.

5.7.4. Element < filter-configuration>

The root element of the XML document with a single child element <options> and a single

child element <filters>.

5.7.5. Element <options>

The element defines global settings that apply to all DN attribute searches.

sizeLimit – The size limit for the search operations. Default: The size limit configured for the domain object.

timeLimit – The time limit for the search operations. Default: The time limit configured for the domain object.

accessPoliciesEnabled – Whether to apply access policies to the search results. Default: **true**.

isPresentSearchEnabled – Whether the searches support match rule isPresent. Deafult: **false**.

5.7.6. Element <filters>

The element serves to group a set of <filter> elements.

5.7.7. Element <filter>

The element maps the name of a search panel attribute and optionally the object set name to search parameters. The search parameters are specified in one or more child elements <search>. Each child element describes a search operation. The total search result is the sum all search results.

additionalFilter – A filter component that must be matched in addition to the filter components resulting from the search elements.

attributes – The comma-separated list of attribute names the filter applies to. Required.

objectSets – The comma-separated list of object sets the filter applies to.If left unspecified, the filter applies to all object sets without specific filter assignment.

5.7.8. Element <search>

The element defines the base object and a search filter.The search filter is applied in addition to the one entered by the user in the search panel.

base – The search base, without domain root.The domain root will be appended.Default: domain root.

filter – The search filter.Default: (objectclass=*).

5.8. Validators

You can configure validators using standard Struts configuration for validation of user

input.For details see the Struts overview documentation.

You must configure the Validator Plug-In in the Struts configuration file **WEB-INF/config/identity/struts-config.xml**.The validator by default expects the validation rules in file **validator-rules.xml**, but you can pass a list of file names to its property “pathnames”.Here is an example:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
                  value="/WEB-INF/config/validator-rules.xml,
                         /WEB-INF/config/validation.xml"/>
</plug-in>
```

This definition instructs the Validator Plug-In to read its validation rules from the files “validator-rules.xml” and “validation.xml”. This approach allows you to separate validator definitions from validation forms to which the validations are applied. As a result, you can re-use your validators in another application and define the validations that are specific to each application.

According to the Struts documentation, your forms must extend the class **org.apache.struts.validator.action.ValidatorForm** instead of **org.apache.struts.action.ActionForm**.

Then, when the validate method is called, the action’s name attribute from the Struts Configuration is used to load the validations for the current form. The form element’s name attribute in the Validator configuration should match the action element’s name attribute.

The Struts Validator comes bundled with several ready-to-use validators. They include: required, mask, byte, short, int, long, float, double, date (without locale support), and a numeric range.

Web Center is prepared for validations. Most of its forms are implemented by the class **com.siemens.webMgr.model.DynaLocaleForm**, which extends the Struts class **DynaValidatorActionForm** and in turn the **ValidatorForm**.The Struts configuration file “struts-config-xml” in folder “config/identity” defines the Validator plug-in among a number of other plug-ins. In order to activate validations for one of the forms, you must insert the proper definitions in the file “validations.xml”.

Note that Web Center support for validations is incomplete in that way that the Struts handling of form validation errors does not really fit together with the error handling in Web Center.

The Struts Validator Plug-In is by default disabled for security reasons. You can enable it via property **struts.validator.enabled** in file **webCenter.properties**.

6. Identity API

The Identity API is used by Web Center to access the service layer implementing the DirX Identity business logic and to access the LDAP layer for searching and modifying the objects in the data store.

The Identity API hides the details of the service layer implementation. It references the business objects by identifiers (being usually the object's distinguished name) and uses standard data types whenever this is possible.

The API methods are available in the JSPs to implement own or modify existing actions in Web Center. Furthermore, the API may be used in a standalone Java application for DirX Identity.

The API methods are grouped into several interfaces. All interfaces and their implementation are available from the jar files that come with the Web Center installation.

6.1. Interface Summary

The Identity API offers the following interfaces:

Name	Description
Approval	Defines methods providing DirX Identity approval functionality.
Assignment	Defines methods providing DirX Identity privilege assignment functionality.
AuthenticatedUser	Defines the methods that have to be implemented by external authentication mechanisms.
Certification	Provides access to certification campaigns.
Delegation	Defines methods providing the old DirX Identity delegation functionality.
Delegations	Defines methods providing the new DirX Identity delegation functionality.
Objects	Defines general object management methods.
RequestWorkflow	Defines request workflow management and access methods.
Session	Defines methods providing session-related functionality.
Users	Extends the Objects interface and defines special user management methods.

For details, refer to the javadoc provided with Web Center in the docs-api folder.

The use of the interfaces Session, Assignment, Objects and Users is demonstrated in a sample program.See the section "Sample application UserSample.java" in this chapter.

6.2. Basic Concepts

This section provides information about basic concepts.

6.2.1. Technical Users and Effective Users

The API distinguishes between a technical and an effective user.The technical user provides the LDAP bind to the directory (should be the DomainAdmin of the particular domain).The effective user must exist in the directory, too.In the Web Center login process, the effective user's credentials are verified, and the user object is loaded from the directory.The technical user, however, always owns the bind to the directory.

6.2.2. Access Control

The access rights are controlled by DirX Identity Access Policies that are checked in the service layer which is part of the Identity API.These access policies are evaluated with respect to the logged-in (effective) user.Since each user accesses the directory with the technical bind, the DirX Identity algorithms are not affected by the LDAP ACLs of the effective user.

6.2.3. Session Handling

Each user working with the API must be provided with his own API session. In Web Center, a new session is created each time a new user logs in. The API session object is held as an attribute of the HTTP-session, thus each HTTP session has its own unique API session. The session manages the LDAP connection, access control, and access to the business logic. It is implicitly contained in the implementation of each interface.

6.2.4. Business Logic

Since Web Center uses the DirX Identity API to access the Data Store, all aspects of DirX Identity Business Logic available in Web Center are covered by the use of the API, too. Note, that among others, the following features are available via the API:

Feature	Interface.method
Triggering an offline role resolution	Objects.saveObject
Check for SoD violation	Objects.saveObject
Start approval workflows in case a new privilege is assigned that must be approved	Objects.saveObject
Evaluate Object Description during modification of a property	Objects.setObjectProperty
Evaluate Object Description during save of the object.	Objects.saveObject
Create default property values by evaluating the object descriptions when a new object is created.	Objects.createObject

Evaluation of ObjectDescriptions is performed with full functionality as in

WebCenter/Identity Manager, since the same algorithms of the underlying layers are used.This includes the evaluation of naming rules and JavaScripts.

6.3. Special Aspects

Some special aspects of the API methods are discussed here in more detail.

6.3.1. Data Conversion

The Objects interface offers the methods getObjectProperty and setObjectProperty to read and modify an object's data.

Both methods load the object from the data store to the storage layer cache, in case the object is not already cached.Note that the data type returned by getObjectProperty and accepted by setObjectProperty is related to the object description assigned to the considered object.Some types defined in the object descriptions are converted to more general, simpler types, when getObjectProperty is called.The following table shows the mapping of object description types to simple types:

Simple type	Object description type
java.util.Date	siemens.dxm.util.GeneralizedTime
java.util.String	com.siemens.idm.server.config. nodes.IDMTimeintervalImpl
java.util.String	siemens.dxm.storage.StorageObject
java.util.String[]	siemens.dxm.storage.StorageObject[]
java.util.String	siemens.dxr.service.nodes.SvcNode
com.siemens.webMgr. identityAPI.Parameter	siemens.dxr.service.types. AssignmentRPValue[]
java.util.String[]	siemens.dxr.service.types.RoleMatchRule[]
java.util.String[]	siemens.dxr.service.nodes.SvcRoleParam[]

Examples:

The date attributes dxrStartDate, dxrEndDate are defined as GeneralizedTime in the user's object description User.xml. A call to Objects.getObjectProperty(dn, "dxrStartDate") returns a Java Date object. Note that in the forms-config.xml file, the type java.util.Date must be used for such date attributes. If a date must be provided in Objects.setObjectProperty, a Java Date object must be be passed as value.

A user's manager is defined as StorageObject in the User.xml object description. A call to Objects.getObjectProperty(dn, "manager") returns a String containing the manager's dn rather than a StorageObject.
Vice versa, Objects.setObjectProperty expects a String containing the manager's dn as value.

6.3.2. Filtering Assigned Privileges

The identity API contains the method listObjects in the Objects interface.

If assigned objects are requested (assignedRoles, assignedPermissions, assignedGroups, or assignedAccounts) the filter and base parameters are evaluated.This is intended to speed up displaying assignments to selected privileges in situations where users possess many privileges.

The following hints have to be considered:

- Set filter = null or base = null to get all assigned privileges.
- If you want to read a certain assigned privilege, use filter="(objectClass=*)" and the privilege's DN as base.The filter "(objectClass=*)" is evaluated with best performance (always match).It is preferable over "(objectClass=dxrRole)" etc.
- Be aware that the assignments returned by the search may be only used for read/display.If you want to select an assignment and to update it later, you have to search the assignments for the given type using filter = null or base = null and identify the assignments to be modified on this basis.The reason is, that during save all new assignments are compared to all old assignments.Assignments being read by a filter are ignored for update.
- If you have already developed some code with Identity API, check the listObjects calls for assigned privileges.Set filter = null and base = null if you do not intend filtering here or if you want to update the assignments afterwards.

6.4. How Web Center Uses the API

This section describes how Web Center uses the API.

6.4.1. How the API is Used in Initialization

After Tomcat starts, the first login request to Web Center triggers the following actions:

- Create a new API session
- Bind to the directory as technical user
- Load the object descriptions
- Authenticate the effective user
- Pass the object descriptions to the API session.

After successful login, the Web Center support object is initialized and is ready for use.

6.4.2. How the API is Used in JSPs

The API is called from the JSPs implementing the actions. Using the Web Center taglibs rather than directly programming on the API accesses the standard Web Center functionality. The taglibs are implemented on top of the API. In special cases, the API is accessed directly by inserting Java code into the JSP.

The following steps are needed to use the API in a Web Center JSP:

1. Import the DataUtils class and the appropriate interfaces to the JSP. See section “DataUtils.java” in chapter “Utility Classes” for details on the DataUtils class.
2. Get an instance of the required interface by calling the appropriate getter for the support object from the DataUtils.
3. Call the API methods.

As a sample, look at the **JSP WEB-INF/jsp/controller/tasks/setGroupForward.jsp**. Here are some excerpts of that JSP. Note, that session is a predefined variable in a JSP holding the HTTP session.

```
...
1) <%@ page import="com.siemens.webMgr.identityAPI.Objects,
             com.siemens.webMgr.model.DataUtils,
...

2) <% Objects objects = DataUtils.getObjectSupport(session);
3) forward = (String)objects.getObjectProperty(groupDN,
       "targetsystem.dxrOptions(dialog)");
... %>
```

6.5. Using the API in a Standalone Application

This section describes how a standalone application uses the API.

6.5.1. Using the API in Initialization

If a standalone application uses the API, the initialization steps described in section “How the Web Center Uses the API” must be coded explicitly.

The following sequence of API calls is used:

ServiceSupport.init	**Initializes the API. It expects data for the system bind (technicalUser and technicalPassword) as well as a property mapper. The mapped item is expected to be an array of two strings containing the mapped type and the relation. This method must be be called only once in the application. It does not trigger any bind / connection but just stores the data in static variables for later use in startSession.**
Session.startSession	Creates a new directory session. This action must be performed for each new user logging in to the application. The configuration will be loaded only once, thus the second and all subsequent calls are much faster than the first call.

The interface of the ServiceSupport.init method is described here in more detail:

```
void init (java.util.Map[] objectConfigs, java.util.Map[]
relationConfigs, java.util.Map rendererMapping, java.lang.String
technicalUser, java.lang.String technicalPassword, java.lang.String
passwordMode);
```

The first argument objectConfigs expects a Map array that contains the information of Web Center's objects-config.xml file. Note that each Map in the array contains the information of one <object> element in the configuration file. The attribute names are mapped to the key, the values are stored as data of the Map.

The following table shows the mapping from the Map keys to the objects-config.xml attribute names:

Key	Mapping
name	Value of id attribute
setName	Value of set attribute. If this does not exist, value of id attribute + "s"
classInDatabase	Value of dirClass attribute
masterClass	Value of masterClass attribute
objectClass	Value of nodeClass attribute

Here is a programming sample for the "user" entry in the objects-config.xml:

User entry in objects-config.xml:

<object id="user" dirClass="dxrUser" nodeClass="siemens.dxr.service.nodes.SvcUser"/>

Code for populating the corresponding Map:

```
Map[] maps = new Map[23];
maps[0] = new Hashtable();
maps[0].put("name", "user");      // from id attribute
maps[0].put("setName", "users"); // the id attribute + "s"
maps[0].put("classInDatabase", "dxrUser"); // from dirClass
maps[0].put("objectClass", "siemens.dxr.service.nodes.SvcUser");
// from nodeClass

maps[1] = new Hashtable();
...
```

The second argument `relationConfigs` expects a Map array that contains the information

of Web Center's objects-config.xml file. Note that each Map in the array contains the information of one <relation> element in the file.

For each relation element, one Map is added to the array with two entries. The entry with key "name" contains the value of the name attribute, the entry with key "mappedTo" contains the value of the "value" attribute:

Entry in objects-config.xml:

<relation name="all" value=".all"/>

Code for population the corresponding Map:

```
Map[] relations = new Map[8];
Hashtable relation = new Hashtable();
relations[0] = relation;
relation.put("name", "all");
relation.put("mappedTo", ".all");
relation = new Hashtable();
relations[1] = relation;
...
```

As an easy alternative, you may let the utilitity class samples.util.ObjectsConfig generate the two Map arrays from an objects-config.xml file. This way, you don't have to hardcode the maps into your program, but can simply use the objects-config.xml file delivered with Web Center. Just create an ObjectsConfig object, passing the path name of the objects-config.xml file as a parameter. Then call getObjectsConfig on the new object:

```
Map[][] objectsAndRelations = oc.getObjectsConfig();
if (objectsAndRelations != null) {
    Map[] objectsMapArray = objectsAndRelations[0];
    Map[] relationsMapArray = objectsAndRelations[1];
    ...
}
```

The third argument rendererMapping expects a Map containing the default renderer configuration. In a standalone application that does not use the Web Center rendering algorithms, an empty Map is used.

The last three arguments contain the technical user's dn, the password, and the passwordMode, for example

```
ServiceSupport.init(maps, relations, "cn=DomainAdmin,cn=My-Company",
```

```
"dirx", "identity");
```

The values are stored in static variables for later use. No bind or other server interaction is performed within this method.

The interface of the ServiceSupport.startSession method is described here in more detail:

```
int startSession(java.lang.String host,
                 int port,
                 java.lang.String domain,
                 java.lang.String user,
                 java.lang.String password);
```

user must contain the effective user's distinguished name.

Here is a sample call to startSession in the My-Company domain:

```
int rc = session.startSession("localhost", 389, "cn=My-Company",
"cn=Hungs Olivier,o=My-Company,cn=Users,cn=My-Company", "dirx");
```

`startSession` performs the technical bind (if it does not already exist) and authenticates the effective user. The result is stored in variable rc, with the following result codes:

Result Code	Description
BIND_OK	All binds are OK, session is successfully created.
BIND_TECHNICAL_ONLY	The technical bind is OK, but authentication of the effective user failed. Must be repeated with the correct credentials of the effective user.
BIND_NONE	Technical bind failed. Init must be repeated with correct credentials.

6.5.2. Using the API in a Java Application

To use the API in a standalone Java application, the support object implementing the interfaces that are provided by the API must be instantiated. This step can be performed in private members of a class:

```
private MetaRoleSupport support = new MetaRoleSupport();
private Session session = (Session)support;
private Objects objects = (Objects)support;
private Assignment assignment =(Assignment)support;
```

```
private Users users = (Users)support;
```

Next, the API must be initialized and a session must be started:

```
...
MetaRoleSupport.init (objectConfigs, relationConfigs,
rendererMapping, technicalUser, technicalPassword, passwordMode);
...
int rc = session.startSession(host, port, domain, user, password);
```

A call to the API methods then is made with the same syntax as in the JSP environment.(See the section “How the API is Used in JSPs” for details.):

```
String forward = (String)objects.getObjectProperty(groupDN,
       "targetsystem.dxrOptions(dialog)");
```

6.6. Sample Application UserSample.java

The use of the API is demonstrated in a simple sample application UserSample.java.The source code of the sample is provided in Web Center’s api/samples/src/samples/api folder.The sample executes the following tasks:

1. Initializes the API
2. Creates a user in the data store
3. Modifies the user’s attributes
4. Assigns a role to a user
5. Assigns a role with role parameters to a user
6. Removes a role assignment from a user
7. Displays the user’s attributes
8. Searches for users in the data store and lists the result.

The sample uses the following APIs:

1. Session
2. Objects
3. Users
4. Assignment.

6.6.1. Setting Up your IDE for the Sample

Perform the following steps in your Java IDE to set it up for the sample:

- Create a new project for Java 8
- Add the package samples.api to the new project's source folder
- Import the source file UserSample.java to that package
- Add the jarfiles contained in Web Center's **WEB-INF/lib** folder to the project's class path. Note: You don't really need all jarfiles from **WEB-INF/lib**; see file **api/samples/readme.html** for details.

Now you are ready to compile and run the sample in your IDE.

6.6.2. Initializing the API

Initialization is performed in the sample's init method. Since only users, roles, and role assignments are subject to the sample, only three configuration entries are added to the property mapper (maps).

The sample runs in the My-Company domain. The DomainAdmin is used for the technical bind.

The effectiveUser is "Hungs Olivier". His DN is passed to the sample's bind method. Once the effective user has bound, the API is ready for use.

6.6.3. Creating a User

Use mode="CREATE" (in main) to create a new user.

The parent folders that allow creation of a user depend on the effective user's access policies. The method sample.getRootNodes displays all root nodes for user creation. It uses the API method Objects.getRootNodesForCreate.

By this method, the application can check if the logged-in user is allowed to create a new user (if there is at least one node returned), and what are the possible parent folders for user creation.

The user is created in sample.create user. This is performed in the following steps:

No	Method	Description
1	Users.createObject	Creates an object of type "user" with the given RDN under the parentDN in memory. Default values of the object's properties are calculated according to the object descriptions. No save to Ldap is performed at this step.
2	Users.setObjectProperty	Each call adds one property to the user object in memory.

No	Method	Description
3	Users.saveObject	Saves the user to the data store. Internally one Ldap modification set is created, and the user is added with one Ldap ADD operation.

Use Objects.createObject to create other objects in memory. Setting the properties and saving the object is performed as described in steps 2 and 3 above.

Note, that “Olivier Hungs” has no create user access policy. Thus, no user is created, but the following error message is displayed:

Operation not allowed, no access policy found.

Just assign the group UserAdmins of the DirXmetaRole Target System to Olivier Hungs to enable him for user creation. Or use another login.

6.6.4. Modifying an Object

The newly created user is modified using sample.modifyData. This method may be used for all types of objects, not only for users.

Modification is performed in the following steps:

No	Method	Description
1	Objects.setObjectProperty	Checks whether the object is present in the storage layer cache. If not, the object is loaded from the data store. Modifies the cached object’s property to the current value. Note that a value of null removes an attribute. No save to Ldap is performed at this step.
2	Objects.saveObject	Saves the modifications to the data store.

6.6.5. Moving an Object

Moving an object to another parent DN is available in the Objects Interface. The method Objects.moveTo expects the parameters objectId (the DN of the object to be moved) and newParentId (the DN of the object’s new parent).

Note that Objects.moveTo does not check for access policies. You should call Users.isMoveAllowed to consider access policies prior to moving the object.

If you modify the object’s naming attribute by calling Objects.setObjectProperty and Objects.save (see above), the object is renamed in the data store. If you use a DirX Server, all DN changes (rename and move) are performed by the server in each attribute of type DirectoryDN, i.e. DN references to the object are automatically kept during move or rename.

6.6.6. Displaying an Object

Sample.showUserData displays some selected data of the user. It uses Objects.getObjectProperty to retrieve the data.

getObjectProperty first checks whether the object with the requested DN exists in the storage layer cache and loads it on demand. Next, it reads the property value from the object, taking into account the object description.

Note that multivalued properties are returned as an array, as it is shown with objectClass in the sample.

6.6.7. Searching Users

Sample.listUsers searches users in the directory and displays for each returned user the set of requested attributes.

The search is performed by Objects.listObjects. The method searches the directory according to the given filter and search base. It returns a two-dimensional list. The outer list contains one inner list for each object being returned by the search. The inner list contains the requested attributes in the order that was passed to listObjects.

6.6.8. Assigning a Role

To run the sample for assigning roles, change the mode in the main method to mode=”ASSIGN”. In this mode, sample.assignRole is called for the Gold Customer role, and sample.assignRoleWithParameters is called to assign the Project Member role, that requires the role parameter “Project” to be supplied with the assignment.

A role assignment of a role without parameters is performed in the following steps:

No	Method	Description
1	Objects.listObjects	Checks if the role to be assigned is available. This is performed by calling listObjects for “availableRoles”. If a list of length = 0 is returned, then the role is not available for assignment. This may be due to a missing grant policy for the logged-in user.
2	Assignment.assignObjectTo	Assigns the privilege to the user in memory.
3	Users.saveObject	Saves the user and his assignments to the data store. Triggers an offline role resolution.

In case a role with parameters is assigned, the role parameters must be added to the assignment. A sample for this procedure is shown in sample. assignRoleWithParameters. The assignment is performed in the following steps:

No	Method	Description
1	Objects.listObjects	Checks if the role to be assigned is available. This is performed by calling listObjects for “availableRoles”. If a list of length = 0 is returned, then the role is not available for assignment. This may be due to a missing grant policy for the logged-in user.
2	Assignment.assignObjectTo	Assigns the privilege to the user in memory.
3	Objects.listObjects	Lists all assignedRoles of the user. For each assignment, the role’s cn, the assignment’s uid, and the role’s dn is requested. This step is required to get the assignment’s uid.
4	Assignment. hasAssignableParameters	Checks if we have a role with role parameters
5	Assignment. getAssignmentParameters	Returns the array of role parameters that are required for the considered role.
6	aParam.getProposals	Returns the array of possible values for that parameter. In a GUI, the user must select one or more parameter values and add it to the assignment.
7	aParam.setValues	Sets the selected role parameter values in the parameter object
8	Assignment. setAssignmentParameters	Adds the parameter values to the assignment
9	Users.saveObject	Saves the user and his assignments to the data store. Triggers an offline role resolution.

Roles with role parameters may be assigned multiple times to the user. The assignment’s uid serves to identify the assignment in that case.

Multiple assignments (with and without role parameters) may be assigned or removed. All the edits are performed in memory, until the last step users.saveObject is called. A final call to Objects.resetObject (instead of saveObject) undoes all edits.

6.6.9. Removing a Role Assignment

To run the sample for removing role assignments, change the mode in the main method to mode=”UNASSIGN”. In this mode, sample.unAssignRole is called for the Project Member role.

A role assignment is removed in the following steps:

No	Method	Description
1	Objects.listObjects	Lists all roles being assigned to the user
2	Assignment.unassignObjectFrom	Removes the assignment in memory
3	Users.saveObject	Saves the user and his remaining assignments to the data store. Triggers an offline role resolution.

7. Utility Classes

The utility classes are important in the Web Center environment. Their use is not required in a standalone application.

7.1. ActionUtils.java

The class ActionUtils implements commonly used methods for action handling:

Method	Description
getExceptionForward	Returns the exception forward from an exception
getFormConfiguration	Returns the form configuration for the given form name
getForwardFromRequest	Returns the forward name calculated from the request parameters
getParameters	Reads the parameters from an action mapping

Refer to the Javadoc for the correct syntax of the method calls.

7.2. DataUtils.java

The class DataUtils implements commonly used methods for data handling and manipulation.

In the WebCenter JSP environment, the following static getters are important since they return an instance implementing the interfaces provided by the API:

Interface	Method providing an instance of the interface
Approval	getApprovalSupport
Assignment	getAssignmentSupport
Certification	getCertificationSupport
Delegation	getDelegationSupport
Delegations	getDelegationsSupport
Objects	getObjectSupport
RequestWorkflow	getRequestWorkflowSupport
Session	getSessionSupport
Users	getUserSupport

All the methods listed above are called with one parameter, the HTTP session of the request.

Sample call:

```
Objects support = DataUtils.getObjectSupport(session);
```

The following methods are used to read data from the storage layer or to store the data in the storage cache (without saving the object):

Method	Description
fetchDataset	Retrieves the dataset with the given name from the storage
storeSelectedBeanData	Stores the data with the given attribute names in the storage cache
storeBeanData	Stores the bean property data with the given attribute name
storeAllBeanData	Stores all bean property data
storeBeanArrayData	Stores data from a table (e.g. match rules).

Refer to the Javadoc for the correct syntax of the method calls.

fetchDataset returns an array of DirectoryEntryBean. It uses listObjects from the Objects API to retrieve the data and converts the list structure to the bean array.

The following code sample shows how to use storeSelectedBeanData to save the data of a form to the selected user:

```
Objects objects = DataUtils.getObjectSupport(session);

String dn =
    (String)session.getAttribute("com.siemens.webMgr.selectedUser");
DynaLocaleForm form =
    (DynaLocaleForm)session.getAttribute
        ((String)pageContext.getAttribute("formName"));

DynaBean bean = (DynaBean)form;
Map formMap = form.getMap();
String[] fieldnames =
    (String[])formMap.keySet().toArray
        (new String[formMap.keySet().size()]);
DataUtils.storeSelectedBeanData(bean, dn, fieldnames, null, session);
```

7.3. DirectoryEntryBean

The class DirectoryEntryBean implements a DynaBean to hold the properties of a single directory entry.If tables are displayed in webCenter (for example, assigned roles /

permissions / groups, result of a user search, parameters, ...) the values are stored in arrays of DirectoryEntryBean.

Method	Description
initialize	Initializes the entry with default values
assignEntryData	Sets the entry data for this bean. This method is applied during bulk-data reading from the support bean.
get	Returns the value of a simple property with the specified name.
set	Sets the value of a simple property with the specified name.

Refer to the Javadoc for the complete set of methods and for the correct syntax of the method calls.

Example:
Get the DN of a selected object in a JSP (see deleteObjects.jsp)

```
String itemStr = (String)pageContext.getAttribute("item");
DirectoryEntryBean[] dataset =
    (DirectoryEntryBean[])pageContext.getAttribute("dataset");
int num = Integer.parseInt(itemStr);
if (dataset != null && num < dataset.length) {
    Users userSupport = DataUtils.getUserSupport(session);
    String objectId = (String)(String)dataset[num].get("dn");
    ...
}
```

7.4. DNUtilities

The class DNUtilities provides static DN utility methods:

Method	Description
isDN	Checks whether the argument is a distinguished name.
equals	Checks whether two distinguished names are equal.
getLastRDN	Returns the relative distinguished name of a dn.
getParent	Returns the parent DN.
removeFirstRDN	Removes the first RDN from a DN.
makeDN	Builds a DN from type, value, and parent dn.
getNamePart	Returns the value of the RDN component for the given index.
getLastNamePart	Returns the RDN's name part.

Method	Description
isLastRDNType	Checks whether the provided RDN type equals to the last RDN's type.
isWorkflowDN	Checks for a request workflow DN.
makeActivityDN	Builds an activity DN from the workflow instance id and the activity name.
makeWorkflowDN	Builds a workflow DN from the workflow instance id.
getReversedNameParts	Returns the DN's component values in reverse order.

Refer to the Javadoc for the correct syntax of the method calls.

7.5. Message

The static class Message provides utility methods to access the message attributes in the session:

Method	Description
set	Sets a text, parameters and the confirmation flag.
getText	Retrieves the message.
setText	Sets a message text.
getConfirmationFlag	Retrieves the confirmation flag.
setConfirmationFlag	Sets the confirmation flag.
getParameters	Retrieves the message parameters.
setParameters	Sets the message parameters.
clear	Clears the message, the message parameters and the confirmation flag.
save	Returns message, confirmation flag and parameters as an array of Object.
restore	Restores message, confirmation flag and parameters to the session.

Refer to the Javadoc for the correct syntax of the method calls.

7.6. FilterUtilities

The class FilterUtilities provides static utility methods for the handling of LDAP filters:

Method	Description
getFilter	Creates an LDAP filter from attribute name, type of operation and value.

Method	Description
getFilter	Creates an LDAP filter from a list of filters and the conjunction. Returns the default filter for an empty list.
escape	Escapes special characters in a value.
countSignificantChars	Counts the significant characters in a value. Wildcards and white spaces are not counted.

Refer to the Javadoc for the correct syntax of the method calls.

8. Web Center Tag Library

This chapter provides information about the Web Center tag library.

8.1. Certification Tags

This section describes the usage of the DirX Identity Web Center certification tag library.Its tags are intended to be used in action JSPs.

To use a certification tag in a JSP page, include the following tag library directive in the page:

```
<%@ taglib
    uri="http://www.siemens.com/directory/webManager/certification"
    prefix="cert" %>
```

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/webMgrCert_1_0.tld** in the **webManager.jar** file, and cannot be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center application is “cert”.

8.1.1. ResetCampaignState Tag

8.1.1.1. Description:

The tag resets the state of a certification campaign to 'PREPARING' and may also clear its dxrError attribute.

The tag returns the operation’s result:

- ok: The campaign has been reset.
- failed: Resetting the campaign failed.
- disallowed: The authenticated user is not allowed to perform the operation.

8.1.1.2. Usage:

`<cert:resetCampaignState` *`Attributes`*`/>`

8.1.1.3. Attributes:

`campaignId="`*`campaignId`*`"`	The campaign DN (String). [In; required; accepts a JSTL expression.]

`clearLogs="true"|"false"`	Whether to clear the dxrError attribute (boolean). [In; optional; default: “true“; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

8.1.2. ResetEntry Tag

8.1.2.1. Description:

The tag discards any unsaved changes to the currently active certification entry.

8.1.2.2. Usage:

```
<cert:resetEntry/>
```

8.1.3. SaveEntry Tag

8.1.3.1. Description:

The tag saves changes to the currently active certification entry.

The mode attribute defines what is really done:

- storeInMemory: Store the changes in memory but don’t write them to the database.
- parameters: Store changes to role parameters in memory but don’t write them to the database.
- save: Write all unsaved changes to the database.
- saveAndFinish: Write all unsaved changes to the database and then finish the certification entry.

Unsaved changes can be undone by a calling the resetEntry tag.

The tag attributes include:

- subjectDN: The DN of the user whose privilege assignments are certified.
- objectSet: The privilege set to be saved. The supported sets are:
- certifyRoles

- certifyPermissions
- certifyGroups
- automaticallyAssignedRoles
- automaticallyAssignedPermissions
- automaticallyAssignedGroups
- resources: The set of privilege assignments to be certified. The set must correspond to the specified object set. It's the DirectoryEntryBean array underlying the relevant form property.

When storing role parameters, the object set must be "certifyRoles" and the tag attributes must include:

- selectedItem: The index of the role (within the resources array) whose parameters are to be saved.
- assignmentParameters: Any map including the role parameters. The map must contain a mapping for each parameter of the role assignment. The key for a role parameter in the map is the role parameter name. The values can be given as a String array or as an instance of class "com.siemens.webMgr.identityAPI.Parameter". Note that user certification only supports deleting role parameter values, not adding or changing values. Caution: The role parameters are removed from the assignmentParameters map passed as input argument.

The tag returns the operation's result:

- saved: The changes have been stored or saved.
- failed: The changes could not be stored or saved.
- unfinished: The changes have been saved, but the certification entry could not be finished since some privilege assignments have not been accepted or rejected yet (mode "saveAndFinish" only.)

8.1.3.2. Usage:

`<cert:saveEntry` *`Attributes`*`/>`

8.1.3.3. Attributes:

`assignmentParameters="`*`assignmentParameters`*`"`	The assignment parameters (java.util.Map). [In; optional; accepts a JSTL expression.]
`mode="parameters"|"save"|"saveAndFinish"|"storeInMemory"`	The save mode. [In; optional; default: "storeInMemory"; accepts a JSTL expression.]
`objectSet="`*`objectSet`*`"`	The privilege set to be saved (String). [In; required; accepts a JSTL expression.]

`resources="resources"`	The privilege assignments to be certified (DirectoryEntryBean[]). [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`selectedItem="selectedItem"`	The index of the selected role for role parameter changes. [In; optional; accepts a JSTL expression.]
`subjectDN="subjectDN"`	The DN of the subject of the certification entry (String). [In; required; accepts a JSTL expression.]

8.1.4. SaveNotifications Tag

8.1.4.1. Description:

The tag saves the active flags of the notifications of a certification campaign.

The tag returns the operation's result:

- ok: The active flags of the notifications have been saved.
- failed: Saving the notifications failed.

8.1.4.2. Usage:

```
<cert:saveNotifications Attributes/>
```

8.1.4.3. Attributes:

`formName="formName"`	The name of the form with the notification list. [In; optional; accepts a JSTL expression.]
`propertyName="propertyName"`	The name of the form property with the notification list. [In; optional; accepts a JSTL expression.]

`result="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2. Controller Tags

This section describes the usage of the DirX Identity Web Center control tag library.Its tags are intended to be used mainly in action JSPs but some can be used in view JSPs as well.

To use a control tag in a JSP page, include the following tag library directive in the page:

```
<%@ taglib
uri="http://www.siemens.com/directory/webManager/controller"
prefix="ctrl" %>
```

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/webMgrCtrl_1_0.tld** in the **webManager.jar** file, and cannot be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center application is "ctrl".

8.2.1. Assign Tag

8.2.1.1. Description:

The tag performs assignment or unassignment operations like

- Assigning privileges to a user, role or group.
- Assigning access rights to a delegation.
- Assigning resources to an access right.

The tag attributes include:

- The receiving object's DN.
- The set of available objects that might be assigned.
- The set of objects already assigned to the receiving object.
- The operation type that defines whether to

- Assign all available objects.
- Assign selected available objects.
- Unassign all assigned objects.
- Unassign selected assigned objects.
- The selection, if just selected objects are to be assigned or unassigned.

The tag performs the requested assignments one by one. If it encounters a role assignment requiring role parameters, it interrupts processing of assignments and requests values for the role parameters from the caller.

The tag returns:

- The updated list of available items.
- The indexes of the not yet processed selected available items in the updated list.
- The requested parameters for a role assignment.

If the tag requests to un-assign a privilege from a list of users, the un-assignment may fail for one or more users; for example, due to missing access rights. In this case, the tag sets the error message attribute in the HTTP session to the message key "privilegeAssign.warning.notUnassignable". The common names of the users for which the un-assignment has failed are combined into a single string and assigned to the error message parameter attribute in the HTTP session. You can define the delimiter between common names, a prefix preceding the first common name, and a suffix trailing the last common name via tag attributes. Note that the message is usually displayed via Javascript in the browser.

8.2.1.2. Usage:

`<ctrl:assign <i>Attributes</i>/>`

8.2.1.3. Attributes:

`assigned="`*`varName`*`"`	The name of the variable referencing the assigned objects (Object[]). [In; optional; accepts a JSTL expression.]

`assignmentParameters="`*`varName`*`"`	The name of the variable referencing the mapping instance for the assignment parameters (java.util.Map). The map must contain the keys "selectedAvailable" and "selectedAssigned" each of which must point to a comma-separated list of item indexes in the respective object array. The list may be a simple String, or the first String in a String array. The index of the first item is 0. If no item is selected, specify “-1” or just null. The map is updated during assignment and can be used later on for view update purposes. [In/Out; optional; accepts a JSTL expression.]
`available="`*`varName`*`"`	The name of the variable referencing the available objects (Object[]). [In; optional; accepts a JSTL expression.]
`msgNameListDelim="`*`msgNameListDelim`*`"`	The delimiter between privilege names in error messages including a privilege name list. [In; optional; default: “\n- ”; accepts a JSTL expression.]
`msgNameListPrefix="`*`msgNameListPrefix`*`"`	The prefix preceding the first privilege name in error messages including a privilege name list. [In; optional; default: “\n- ”; accepts a JSTL expression.]
`msgNameListSuffix="`*`msgNameListSuffix`*`"`	The suffix following the last privilege name in error messages including a privilege name list. [In; optional; default: no suffix; accepts a JSTL expression.]
`objectId="`*`dn`*`"`	The unique identifier (DN) of the object receiving the assignment. [In; required; accepts a JSTL expression.]
`operation="assignAll"|"assignSelection"|` `"unassignAll"|"unassignSelection"`	The operation to be performed. [In; required; accepts a JSTL expression.]

`scope="page"|"request"|"session"|"application"`	The scope for varNewAvailable, varNewParameters and varNewSelectedAvailable. [In; optional; default: “page“; accepts a JSTL expression.]
`varNewAvailable="`*`varName`*`"`	The name of the variable to accept the new available items (DirectoryEntryBean[]). [Out; optional; default: “newAvailable“; accepts a JSTL expression.]
`varNewParameters="`*`varName`*`"`	The name of the variable to accept the parameters for a role assignment (Parameter[]). [Out; optional; default: “newParameters“; accepts a JSTL expression.]
`varNewSelectedAvailable= "`*`varName`*`"`	The name of the variable to accept the indexes of the new selected available items (String). [Out; optional; default: “newSelectedAvailable“; accepts a JSTL expression.]

8.2.2. AssignPrivilegeToUsers Tag

8.2.2.1. Description:

The tag assigns a privilege to a list of users.

The tag attributes include:

- The DN of the privilege to be assigned.
- The DNs of the users the privilege is to be assigned to.
- Start date, end date, role parameters and re-approval flag for the assignments, if any. The values do not vary with the users.
- A flag that controls operation signing.
- init: Don’t perform any assignment. Just generate and return an XML document to sign the assignments.
- finish: Validate the provided signed document. Perform the assignments in the directory, and save the signed document.
- The due date for the assignment.
- The reason for requesting the assignments. The reason is displayed to potential approvers.

The tag returns:

- The XML document to be signed if requested.

- The operation’s result:
- OK: The operation was successful.
- READ_ERROR: The privilege could not be read.
- DATE_CONFLICT: Start date lies after end date.
- EMPTY_LIST: The user list is empty.
- UNKNOWN_TYPE: The privilege DN denotes neither a role nor a permission nor a group.
- ROLE_PARAMETERS_MISSING: No parameters provided for a role requiring parameters.
- ASSIGNMENT_ERROR: The privilege could not be assigned to one or more users.
- INVALID_SIGNATURE: The provided signature could not be validated.
- SAVE_SIGNATURE_ERROR: The provided signature could not be saved.
- Detailed information in case of assignment errors:
- The DNs of the users the privilege has been assigned to.
- The DNs of the users the privilege could not be assigned to.

The DNs of the users an approval workflow has been started for due to SoD violations.

8.2.2.2. Usage:

`<ctrl:assignPrivilegeToUsers` *`Attributes`*`/>`

8.2.2.3. Attributes:

`dueDate="`*`dueDate`*`"`	The assignment due date. [In; optional; accepts a JSTL expression.]
`endDate="`*`endDate`*`"`	The end date for the assignment. [In; optional; accepts a JSTL expression.]
`errorInfo="`*`varName`*`"`	The name of the variable to accept error details in case of result ASSIGNMENT_ERROR: (Object[]) errorInfo[0]: String[]: User DNs of successful assignments errorInfo[1]: String[]: User DNs of failed assignments [Out; optional; default: “errorInfo”; accepts a JSTL expression.]
`needsReapproval="true"|"false"`	Whether the assignment must be re-approved. [In; optional; default: “false“; accepts a JSTL expression.]

`parameters="parameters"`	The parameters for a role assignment. [In; optional; accepts a JSTL expression.]
`privilege="dn"`	The DN of the privilege. [In; required; accepts a JSTL expression.]
`reason="reason"`	The request reason. [In; optional; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation's result. [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for errorInfo and result; defaults to "page". [Out; optional; default: "page"; accepts a JSTL expression.]
`scopeTextToBeSigned="page"|"request"|"session"| "application"`	The scope of the variable to accept the text to be signed. [Out; optional; default: "page"; accepts a JSTL expression.]
`signedData="signedData"`	The signed XML document. [In; optional; accepts a JSTL expression.]
`signState="init"|"finish"`	Whether to start or finish signing the operation. [In; optional; accepts a JSTL expression.]
`startDate="startDate"`	The start date for the assignment. [In; optional; accepts a JSTL expression.]
`users="dns"`	The DNs of the users the privilege is to be assigned to (String[]). [In; optional; accepts a JSTL expression.]
`varTextToBeSigned="varName"`	The name of the variable to accept the XML document to signed (String). [Out; optional; default: "varTextToBeSigned"; accepts a JSTL expression.]

8.2.3. ChangePassword Tag

8.2.3.1. Description:

The tag performs user password changes. It can be used

- When a user changes his password.
- When an administrator resets a user password to a specific value.
- To generate a random password that complies with a user's password policy. In this case, the user password in the directory is left unchanged.

The tag attributes include:

- The user's DN.
- The new password. If left unspecified, a password is automatically generated; in this case, the attributes login and reset are ignored
- Whether the password is reset by an administrator or changed by the user himself. This impacts changes to various password management attributes.
- Whether the user is regarded as logged in after the password change. This is usually the case if a user has been forced to change his password after a challenge/response authentication.

The tag returns:

- The generated password for a successful password generation.
- The string "saved" for a successful password change.
- The string "invalid" if the provided password did not comply with the user's password policy.
- The string "failed" if the password could not be changed or generated.

The tag can be used the same way for changing account passwords.

8.2.3.2. Usage:

`<ctrl:changePassword` *`Attributes`*`/>`

8.2.3.3. Attributes:

`correlationId="`*`correlationId`*`"`	The correlation identifier to enable password change tracking. [In; optional; accepts a JSTL expression.]
`login="true"|"false"`	Whether the user is regarded as logged in after the password modification. [In; optional; default: "false"; accepts a JSTL expression.]

masterAccounts="*masterAccounts*"	The user's master accounts for password synchronization (DirectoryEntryBean[]). [In; optional; accepts a JSTL expression.]
omitUser="true"|"false"	Whether both user and account passwords are to be changed, or account passwords only. [In; optional; default: "false"; accepts a JSTL expression.]
password="*password*"	The new password. [In; optional; accepts a JSTL expression.]
preventSyncAccountFlag="*preventSyncAccountFlag*"	The name of the form property that defines whether the password should be synchronized to the accounts. [In; optional; accepts a JSTL expression.]
reset="true"|"false"	Whether the password is reset by an administrator, or changed by the user himself. [In; optional; default: "false"; accepts a JSTL expression.]
result="*varName*"	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
scope="page"|"request"|"session"|"application"	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
syncAccounts="*syncAccounts*"	The user's accounts for password synchronization (DirectoryEntryBean[]). [In; optional; accepts a JSTL expression.]
userId="*dn*"	The DN of the user whose password is to be changed. [In; required; accepts a JSTL expression.]

8.2.4. CheckAccessPolicy Tag

8.2.4.1. Description:

The tag checks if the logged in user has the access right

- To perform one or more operations on a given object, or on a set of objects, or

- To select a given menu item.

To check for the right to perform an operation, include the following tag attributes:

- The operations.
- The DNs of the objects for the operation.

To check for the right to perform a menu item, leave the operation unspecified and include the following tag attributes:

- The menu key; for example “SelfService.”
- The menu item; for example: “summary.”

The tag returns:

- Boolean.FALSE if the access right is not granted or the check couldn’t be performed for whatever reason.
- Boolean.TRUE if the access right is granted, or if no operation or item was specified.
- The operation that was denied.

8.2.4.2. Usage:

`<ctrl:checkAccessPolicy` *`Attributes`*`/>`

8.2.4.3. Attributes:

`forward="`*`forward`*`"`	Take additional operations from parameter “operation” of the action referenced by the specified forward of the current action. [In; optional; accepts a JSTL expression.]
`item="`*`item`*`"`	The menu item. [In; optional; accepts a JSTL expression.]
`key="key"`	The menu policy key. [In; optional; accepts a JSTL expression.]
`objectId="`*`dnOrDNs`*`"`	The DN of the object (String) or the DNs of the objects (String[]) to perform the operation on. [In; optional; accepts a JSTL expression.]
`operation="`*`operations`*`"`	The comma-separated list of operations to be checked. [In; optional; accepts a JSTL expression.]

`result="varName"`	The name of the variable to accept the operation's result (Boolean). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`subset="some"|"all"`	Whether to return "true" if performing the operation is allowed on at least one of the objects ("some") or only if it is allowed on objects ("all"). [In; optional; default: "some"; accepts a JSTL expression.]
`varDisallowedOperation="varName"`	The name of the variable to accept the operation that was denied (String). [Out; optional; default: "disallowedOperation"; accepts a JSTL expression.]

8.2.5. CheckChallengeResponse Tag

8.2.5.1. Description:

The tag checks whether a user has correctly responded to challenges presented for authentication.

The tag attributes include:

- The map of challenges presented to the user, and the responses given by the user.

The tag returns a boolean value indicating the operation's result:

- true: The user has responded correctly to the presented challenges.
- false: The user has not responded correctly.

8.2.5.2. Usage:

`<ctrl:checkChallengeResponse Attributes/>`

8.2.5.3. Attributes:

`challengesResponses="challengesResponses"`	The map of challenges and responses (java.util.Map). [In; required; accepts a JSTL expression.]

`locked="varLocked"`	The name of the variable indicating whether the user is currently locked from authenticating via CR (Boolean). [Out; optional; default: “locked“; accepts a JSTL expression.]
`minChallenges="minChallenges"`	The minimum number of challenges to answer correctly. [In; optional; default: see webCenter.properties; accepts a JSTL expression.]
`objectId="dn"`	The DN of the user whose password is to be checked. [In; optional; default: the logged-in user; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation’s result (Boolean). [Out; optional; default: “result“; accepts a JSTL expression.]
`resultMap="resultMap"`	A scoped variable of type Map<String,Boolean>. The map is filled with the check results (challenge -> true if answer ok, challenge -> false otherwise). [Out; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

8.2.6. CheckChallengeResponseCompliance Tag

8.2.6.1. Description:

The tag checks if challenge/response pairs entered by a user are compliant with the following rules:

- The minimum number of pairs is specified. The number can be set via option “minEnteredChallenges” in file **webCenter.properties**.
- All mandatory questions have been answered.
- No response falls short of the minimum response length. The default minimum length is taken from configuration parameter “challengeResponses.minimumResponseLength” in file **webCenter.properties**.
- There are no duplicate responses. This check can be enabled or disabled via option “challengeResponses.duplicateResponsesAllowed” in file **webCenter.properties**.

The tag attributes include:

- The user's DN.
- The challenge/response pairs concatenated to a single string.
- The delimiter between two adjacent challenge/response pairs.
- The delimiter between challenge and response within a pair.
- The minimum number of pairs (if different from the configured value).
- The minimum response length.

The tag returns:

- A string that indicates the operation's result:
- ok: The challenge/response comply with the requirements.
- tooFewChallenges: Too few challenge/response pairs defined.
- invalid: There are duplicate responses, or responses that fall short of the minimum response length, or unanswered mandatory questions.
- failed: Missing DN, or the check could not be performed for some reason.
- In case of result invalid, the invalid pairs are returned in an object of type Map<String,List<String>>. The keys are "mandatory", "minLength" and "duplicates". The values are the questions of the corresponding invalid challenge/response pairs.

8.2.6.2. Usage:

```
<ctrl:checkChallengeResponseCompliance Attributes/>
```

8.2.6.3. Attributes:

`challengesResponses="challengesResponses"`	The challenge/response pairs (String). [In; required; accepts a JSTL expression.]
`crDelim="crDelim"`	The delimiter between challenge and response within a pair. [In; optional; default: "!NV!"; accepts a JSTL expression.]
`json="true"|"false"`	Whether to return the information about invalid CR pairs as a String representing an array of JSON objects, or as a Map. [In; optional; default: "false"; accepts a JSTL expression.]
`minChallenges="minChallenges"`	The minimum number of CR pairs. [In; optional; default: see webCenter.properties; accepts a JSTL expression.]

`minResponseLength="minResponseLength"`	The minimum length of a response. [In; optional; default: see webCenter.properties; accepts a JSTL expression.]
`objectId="dn"`	The DN of the user whose CR pairs are to be checked. [In; optional; default: the logged-in user; accepts a JSTL expression.]
`pairDelim="pairDelim"`	The delimiter between adjacent CR pairs. [In; optional; default: “!NI!”; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`invalidPairs="invalidPairs"`	The name of the variable to accept the invalid pairs (String). [Out; optional; default: “invalidPairs“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

8.2.7. CheckCredentials Tag

8.2.7.1. Description:

The tag checks if a bind to the provisioning database with the given user DN and password succeeds.

The tag attributes include:

- The user’s DN.
- The user’s password.

The tag returns:

- A string that indicates the operation’s result:
- ok: The bind with the given credentials was succesful.
- invalid: The bind failed due to invalid credentials.
- retryLater: Checking the credentials failed for a temporary reason.

- failed: Checking the credentials failed for a non-temporary reason.

8.2.7.2. Usage:

```
<ctrl:checkCredentials Attributes/>
```

8.2.7.3. Attributes:

`password="password"`	The user's password. [In; required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`userId="dn"`	The user's DN. [In; required; accepts a JSTL expression.]

8.2.8. CheckPassword Tag

8.2.8.1. Description:

The tag checks if the given user password complies with the user's password policies.

The tag attributes include:

- The user's DN.
- The password to be checked.

The tag returns:

- A string that indicates the operation's result:
- ok: The password complies with the policies.
- invalid: The password does not comply.
- failed: The password could not be checked, for example, due to a problem when accessing the directory.

8.2.8.2. Usage:

```
<ctrl:checkPassword Attributes/>
```

8.2.8.3. Attributes:

`password="password"`	The password. [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`userId="dn"`	The DN of the user whose password is to be checked. [In; required; accepts a JSTL expression.]

8.2.9. ClearForm Tag

8.2.9.1. Description:

The tag clears the property map of a DynaActionForm. For a DynaLocaleForm, the modification flag is reset as well.

The tag attributes include:

- The form to be reset.

The tag does not return any result.

8.2.9.2. Usage:

```
<ctrl:clearForm Attributes/>
```

8.2.9.3. Attributes:

`form="form"`	The form to be reset (DynaActionForm). [In; required; accepts a JSTL expression.]

8.2.10. CopyPrivileges Tag

8.2.10.1. Description:

The tag copies a user's privileges to another user.

The tag attributes include:

- The DN of the user whose privileges to copy.
- The DN of the user to assign the privileges to.

The tag returns

- The string “true” in case at least one privilege was copied.
- The string “false” in case no privilege was copied.

8.2.10.2. Usage:

```
<ctrl:copyPrivileges Attributes/>
```

8.2.10.3. Attributes:

`result="varName"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`sourceDN="dn"`	The DN of the privilege source user. [In; required; accepts a JSTL expression.]
`targetDN="dn"`	The DN of the privilege target user. [In; required; accepts a JSTL expression.]

8.2.11. Create Tag

8.2.11.1. Description:

The tag creates a new object in the directory.

The tag attributes include:

- The new object’s RDN.
- The DN of the parent object; can be empty for special objects like password policies.
- The type of the new object; for example, “user” or “dxrPwdPolicyCheck”.
- The form from which to get the new object’s attributes.
- A comma-separated list of properties to copy from the form to the new object. If missing, all appropriate attributes are copied. For example, the list “sn,givenName,description” causes all form properties other than sn, givenName and description to be ignored. Each attribute whose name is prefixed with an exclamation mark ("!") is ignored.

The tag returns

- The new object's DN in case the operation was successful.
- An empty result otherwise.

8.2.11.2. Usage:

```
<ctrl:create Attributes/>
```

8.2.11.3. Attributes:

`attributes="attributes"`	An attribute list. [In; optional; accepts a JSTL expression.]
`objectId="dn"`	The DN of the new object's parent; may be empty for special objects. [In; required; accepts a JSTL expression.]
`rdn="rdn"`	The RDN of the new object. [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`source="form"`	The form being the source for the new object's attributes (DynaActionForm) [In; required; accepts a JSTL expression.]
`storeReadOnlyAttributes="true"|"false"`	Whether to store attributes that are flagged as read-only in the form configuration. [In; optional; default: "false"; accepts a JSTL expression.]
`type="type"`	The type of the new object. [In; required; accepts a JSTL expression.]

8.2.12. CreateDelegation Tag

This tag applies to the new delegation implementation only.

8.2.12.1. Description:

The tag creates a new delegation.

The tag attributes include:

- The DN of the delegator.
- The DN of the substitute. The substitute must be different from the delegator.
- The delegation operation, one of “grant” or “approve”.
- The form bean to get the new delegation’s attributes from.
- The list of the names of the attributes to be copied from the form bean to the new delegation. Attributes “dxrAssignFrom“, “dxrAssignTo“ and “dxrOperationImp“ are ignored.
- Whether to ignore warnings about delegation cycles caused by the new delegation.

The tag returns

- The new delegation’s DN in case the operation was successful. An empty result otherwise.
- A string that indicates the operation’s result:
- ok: The operation was successful.
- delegationsDisabled: The new delegations are not enabled.
- circularDelegationChain: The operation failed since the new delegation would lead to a circular delegation chain.
- invalidSubstitute: The substitute coincides with the delegator.
- dateConflict: The new delegation’s start and end date are inconsistent.
- failed: The operation failed for another reason.

8.2.12.2. Usage:

```
<ctrl:createDelegation Attributes/>
```

8.2.12.3. Attributes:

`attributes="attributes"`	The list of additional attribute names for the new delegation. [In; optional; accepts a JSTL expression.]
`delegator="dn"`	The DN of the delegator. [In; required; accepts a JSTL expression.]

`ignoreWarnings="true"|"false"`	Whether to ignore circular delegation warnings (true), or not (false). [In; optional; default: “false”; accepts a JSTL expression.]
`operation="approve"|"grant"|`	The delegation operation. [In; required; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`source="`*`form`*`"`	The form being the source for the new delegation attributes (DynaActionForm) [In; required; accepts a JSTL expression.]
`substitute="`*`dn`*`"`	The DN of the substitute. [In; required; accepts a JSTL expression.]
`varDN="`*`varName`*`"`	The name of the variable to accept the DN of the new delegation (String). [Out; optional; default: “delegationDN“; accepts a JSTL expression.]

8.2.13. Delegation Tag

This tag applies to the old delegation implementation only.

8.2.13.1. Description:

The tag creates or deletes delegation assignments.

When creating a delegation, the tag attributes include:

- DN of the delegable delegation (required).
- The DN of the delegating user and the substitute (both required).
- The operation type “create”.

The tag returns

- The granted delegation’s DN in case the operation was successful.

- The string “FAIL” in case of error.

When deleting a delegation, the tag attributes include:

- DN of the granted delegation (required).
- The operation type “delete”.

The tag returns

- The string “OK” in case of success.
- The string “FAIL” in case of error.

8.2.13.2. Usage:

```
<ctrl:delegation Attributes/>
```

8.2.13.3. Attributes:

`delegatee="`*dn*`"`	The DN of the substitute. [In; optional; accepts a JSTL expression.]
`delegation="`*dn*`"`	The DN of the delegation object. [In; required; accepts a JSTL expression.]
`delegator="`*dn*`"`	The DN of the user delegating access rights. [In; optional; accepts a JSTL expression.]
`operation="create"|"delete"`	The operation to be performed. [In; required; accepts a JSTL expression.]
`result="`*varName*`"`	The name of the variable to accept the operation’s result (String). [Out; optional; no default; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

8.2.14. Delete Tag

8.2.14.1. Description:

The tag deletes one or more objects, access rights permitting.

The tag attributes for deleting a single object include:

- The DN of the object to delete.
- The deletion due date.

The tag returns

- The string “ok” in case of success.
- The string “disallowed” in case of insufficient access rights.
- The string “failed” in case of error.

The tag attributes for deleting multiple objects include:

- The DNs of the objects to delete.

The tag returns

- The string “ok” in case of success.
- The string “failed” in case of errors of insufficient access rights. In this case, the tag also returns which objects have been deleted, which objects could not be deleted due to insufficient access rights and which ones failed to be deleted, and the list of object for which a deletion workflow has already been started before.

If no object is specified the tag returns “ok”.

8.2.14.2. Usage:

`<ctrl:delete *Attributes*/>`

8.2.14.3. Attributes:

`dueDate="due*Date*"`	The deletion due date. [In; optional; accepts a JSTL expression.]
`objectId="*dn*"`	The DN of the object to be deleted (String). [In; optional; accepts a JSTL expression.]
`objectIds="*dns*"`	The DNs of the objects to be deleted (String[]) [In; optional; accepts a JSTL expression.]
`result="*varName*"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]

`varDeleted="varName"`	In case a request for multiple deletions failed only: The name of the variable to accept the DNs of the entries that have been deleted (String[]). [Out; optional; default: “deleted“; accepts a JSTL expression.]
`varDisallowed="varName"`	In case a request for multiple deletions failed only: The name of the variable to accept the DNs of the entries that could not be deleted due to insufficient access rights (String[]). [Out; optional; default: “disallowed“; accepts a JSTL expression.]
`varFailed="varName"`	In case a request for multiple deletions failed only: The name of the variable to accept the DNs of the entries that could not be deleted due to some error (String[]). [Out; optional; default: “failed“; accepts a JSTL expression.]
`varInProgress="varName"`	In case a request for multiple deletions failed only: The name of the variable to accept the DNs for which a deletion workflow is already running (String[]). [Out; optional; default: “inProgess“; accepts a JSTL expression.]

8.2.15. DN Tag

8.2.15.1. Description:

The tag provides some utility methods to handle distinguished names.

The tag attributes include:

- The DN arguments for the method.
- The method name.

The method equals requires two arguments and returns Boolean.TRUE if the first DN represents the same entry as the second DN, and Boolean.FALSE otherwise.

The method getParent accepts a single DN and returns its parent DN (String). If the DN is invalid or doesn't have a parent DN, the result variable is removed from its scope.

The method getRDNs accepts a single DN and returns its RDN values (String[]). If the DN is invalid, the result variable is removed from its scope. The method isAncestor requires two arguments and returns Boolean.TRUE if the second DN is an ancestor of the first DN, and Boolean.FALSE otherwise.

The method isDN accepts a single argument and returns Boolean.TRUE if the argument is a syntactically correct distinguished name and Boolean.FALSE otherwise.

The method isParent requires two arguments and returns Boolean.TRUE if the second DN is the parent of the first DN, and Boolean.FALSE otherwise.

8.2.15.2. Usage:

`<ctrl:dn *Attributes*/>`

8.2.15.3. Attributes:

`dn="*dn*"`	The first DN argument for the method (String). [In; required; accepts a JSTL expression.]
`otherDN="*dn*"`	A second DN argument for the method (String) [In; optional/required depending on method; accepts a JSTL expression.]
`method="equals"|"getParent"|"getRDNs"|"isAncestor"|"isDN"|"isParent"`	The method to invoke (String). [In; required; accepts a JSTL expression.]
`result="*varName*"`	The name of the variable to accept the operation's result (String or Boolean). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2.16. ExchangeUserPersona Tag

8.2.16.1. Description:

The tag exchanges a persona and its associated user..

8.2.16.2. Usage:

`<ctrl:exchangeUserPersona *Attributes*/>`

8.2.16.3. Attributes:

`JavaScriptId="`*`dn`*`"`	The DN of the Javascript being called to perform the exchange. [In; required; accepts a JSTL expression.]
`objectId="`*`dn`*`"`	The DN of the persona to be exchanged. [In; required; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation's result (String or Boolean). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2.17. Export Tag

8.2.17.1. Description:

The tag writes an object list to a temporary file on the server intended to be downloaded to the client. The object list must be available in a property of type DirectoryEntryBean[] of a session-scoped form bean.

The tag supports the export formats HTML, XML or CSV. The HTML format is designed for import into Excel.

8.2.17.2. Usage:

`<ctrl:export *Attributes*/>`

8.2.17.3. Attributes:

`fileType="html"|"xml"|"csv"`	The export format. [In; optional; default: "html"; accepts a JSTL expression.]
`delim="`*`delim`*`"`	The CSV delimiter. [In; optional; default: ";"; accepts a JSTL expression.]

`formName="formName"`	The name of the session-scoped form holding the object list (String). [In; required; accepts a JSTL expression.]
`listProperty="listProperty"`	The name of the form property (of type DirectoryEntryBean[]) holding the object list (String). [In; required; accepts a JSTL expression.]
`namePrefix="namePrefix"`	The file name prefix in the server's temporary working directory (String). [Out; optional; default: "File_"; accepts a JSTL expression.]
`nbsp="true"|"false"`	Whether to print a non blanking space for empty values in HTML tables (Boolean). [In; optional; default: true; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the absolute path name of the temporary export file (String). In case of errors, the variable is removed from its scope. [Out; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2.18. Expression Tag

8.2.18.1. Description:

The tag recursively evaluates a JSTL expression as long as the result is of type String and includes a JSTL expression. The final result is either returned in a scoped variable, or turned into a String (by applying the toString method) and printed to the response writer of the JSP page.

The tag attributes include:

- The expression to be evaluated.
- The expected Java class of the final result.
- The maximum number of times the expression is recursively evaluated. Expression evaluation stops if the result of the previous evaluation step doesn't contain any expressions.
- The default value if the expression evaluates to null or to an empty String. If left

unspecified, the result will be null or the empty String.

- The result variable's name and scope. If the name is left unspecified, the result is handed to the response writer.

8.2.18.2. Usage:

```
<ctrl:expression Attributes/>
```

8.2.18.3. Attributes:

`default="default"`	The default value. [In; optional; accepts a JSTL expression.]
`iterations="iterations"`	The maximum number of times the expression is recursively evaluated. [In; optional; default: "10"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`type="className"`	The expected Java class of the evaluation result. [In; optional; default: "java.lang.String"; accepts a JSTL expression.]
`value="value"`	The expression to be evaluated. [In; required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the operation's result. [Out; optional; accepts a JSTL expression.]

8.2.19. ExtractEntries Tag

8.2.19.1. Description:

The tag extracts a subset from an entry list (for example from a DirectoryEntryBean[]).The entries in the subset are selected via their indexes. The subset contains either the entire entries or their DNs only.

The source list must be specified either directly or as property of a form bean. If the source list is left unspecified, or if the list is empty, the result variable is removed from its scope.

The subset of entries to return may be specified either by their indexes in the list, or by their DNs. If the subset is left unspecified, or if denotes an empty sub list, the result variable is removed from its scope.

The tag throws a JSP exception in case of errors.

8.2.19.2. Usage:

```
<ctrl:extractEntries Attributes/>
```

8.2.19.3. Attributes:

`formName="formName"`	The form bean name. [In; optional; accepts a JSTL expression.]
`propertyName="propertyName"`	The form property name. [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for variable var. [In; optional; default: “page“; accepts a JSTL expression.]
`sort="true"|"false"`	Whether to sort the entries (true), or not (false). Only if list source is a form-property. [In; optional; default “false”; accepts a JSTL expression.]
`source="source"`	The source entry list (a DirectoryEntryBean[]). [In; optional; accepts a JSTL expression.]
`subset="subset"`	The indexes of the entries to return: either an Integer[], or a string with a comma-separated list of indexes, or the string "all" for all entries. Any indexes pointing outside the list are ignored. Takes precedence over subsetDNs. [In; optional; accepts a JSTL expression.]
`subsetDNs="dns"`	The DNs of the entries to return: either a String[], or a single DN as a String. [In; optional; accepts a JSTL expression.]
`type="dn"|"entry"`	The result type; dn: return DNs only; entry: return entries. [In; optional; default: “entry“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the DNs (String[]) or entries (DirectoryEntryBean[]). [Out; required; accepts a JSTL expression.]

8.2.20. ForwardDelegation Tag

This tag applies to the new delegation implementation only.

8.2.20.1. Description:

The tag forwards a delegation to another substitute. It copies the original delegation to a new one with the new substitute. The delegation operation cannot be changed.

The tag attributes include:

- The DN of the delegation to be forwarded.
- The DN of the new substitute. The substitute must be different from the original delegator and from the original substitute.
- The form bean to get the new delegation's attributes from.
- The list of the names of the attributes to be copied from the form bean to the new delegation. Attributes "dxrAssignFrom", "dxrAssignTo" and "dxrOperationImp" are ignored.
- Whether to ignore warnings about delegation cycles caused by the new delegation.

The tag returns

- The new delegation's DN in case the operation was successful. An empty result otherwise.
- A string that indicates the operation's result:
- ok: The operation was successful.
- delegationsDisabled: The new delegations are not enabled.
- circularDelegationChain: The operation failed since the new delegation would lead to a circular delegation chain.
- invalidSubstitute: The new substitute coincides with the original delegator or the original substitute.
- forwardingNotPermitted: The original delegation doesn't permit forwarding.
- dateConflict: The new delegation's start and end date are inconsistent.
- failed: The operation failed for another reason.

8.2.20.2. Usage:

`<ctrl:forwardDelegation` *`Attributes`*`/>`

8.2.20.3. Attributes:

`attributes="`*`attributes`*`"`	The list of additional attribute names for the new delegation. [In; optional; accepts a JSTL expression.]

`ignoreWarnings="true"|"false"`	Whether to ignore circular delegation warnings (true), or not (false). [In; optional; default: “false”; accepts a JSTL expression.]
`objectId="`*`dn`*`"`	The DN of the original delegation. [In; required; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`source="`*`form`*`"`	The form being the source for the new delegation attributes (DynaActionForm) [In; required; accepts a JSTL expression.]
`substitute="`*`dn`*`"`	The DN of the new substitute. [In; required; accepts a JSTL expression.]
`varDN="`*`varName`*`"`	The name of the variable to accept the DN of the new delegation (String). [Out; optional; default: “delegationDN“; accepts a JSTL expression.]

8.2.21. GenerateReport Tag

8.2.21.1. Description:

The tag generates a report for a set of objects.

The tag attributes include:

- The report DN (required).
- The list of accepted report types.
- The DNs of the objects to be handled by the report or search base, scope and filter for searching the objects.
- The prefix for report file names.

The tag returns

- The type of the generated report.

- The String “unsupportedTypes” if none of the accepted types is supported by the report.

8.2.21.2. Usage:

```
<ctrl:generateReport Attributes/>
```

8.2.21.3. Attributes:

`namePrefix="namePrefix"`	The prefix for report file names. [In; optional; default: “Report_“; accepts a JSTL expression.]
`objectIds="dns"`	The DNs of the entries to be handled by the report (String[]). Takes precedence over searchBase, SearchScope and searchFilter. [In; optional/required; accepts a JSTL expression.]
`reportId="dn"`	The DN of the report to generate. [In; required; accepts a JSTL expression.]
`reportTypes="types"`	A comma-separated list of report types accepted by the caller. [In; optional; default: “HTML“; accepts a JSTL expression.]
`searchBase="dn"`	The search base for the entries handled by the report. [In; optional; accepts a JSTL expression.]
`searchFilter="filter"`	The search filter. [In; optional; accepts a JSTL expression.]
`searchScope="basenode"|"onelevel"|"subtree"|"`	The search scope. [In; optional; default:"basenode"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the report type. [Out; optional; default: “reportType”; accepts a JSTL expression.]

8.2.22. Get Tag

8.2.22.1. Description:

The tag is used to read attributes of an object, or to fill a form bean with attributes of an object, or to fill a table property of a form bean with attributes from a search result.

In case the target object is a Map, an Object array or a String, the only relevant tag attributes are objectId, attributes, target and scope.

- If the target object is a Map, the tag reads the attributes of the specified object. For each attribute, it adds an entry to the map with the attribute name as key and the attribute value as value. Attributes with no value are skipped.
- If the target object is an Object array, the length of the array must match the number of attributes to read. The tag reads the attributes of the specified object. The value of the n^{th} attribute is stored in the n^{th} element of the Object array.
- If the target object is a String, the tag reads only the first attribute of the specified object. The attribute value is stored in a scoped-variable with the target string as name. If the attribute has no value or no attribute is specified at all, the variable is removed from its scope.

In case the target object is a form bean, the tag is used to fill some or all of the form's properties with values from the database:

- The DN of the entry to be read, or of the master entry of a search (for example to fill a list form property like assigned roles of a user).
- The form property names defining which form fields to set. If the list is empty, all properties are set. Property names prefixed with an exclamation mark ("!") are ignored.
- A search base for the search operation. If left unspecified, the search base is taken from the request-scoped variable "defaultSearchBase".
- A filter for the search operation. If left unspecified, the filter is taken from the request-scoped variable "defaultFilter".
- An additional filter for searches for request workflow instances only. The filter may for example restrict the search to workflows with a given state, or to those whose end dates fall within the last 14 days.

The tag returns

- The string "ok" in the object with the given DN exists.
- The string "objectNotFound" in case the object doesn't exist or is not readable.

8.2.22.2. Usage:

```
<ctrl:get Attributes/>
```

8.2.22.3. Attributes:

`attributes="attributes"`	The comma-separated list of attributes or form properties to be returned. [In; optional; accepts a JSTL expression.]
`filter="filter"`	The search filter. [In; optional; accepts a JSTL expression.]
`objectId="dn"`	The DN of the entry to be read, or of the master object for a search operation. [In; optional; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the result (String). [Out; optional; default: “result; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the target variable. [In; optional; default: “page“; accepts a JSTL expression.]
`searchBase="dn"`	The search base. [In; optional; accepts a JSTL expression.]
`target="target"`	The target object. [Out; required; accepts a JSTL expression.]
`workflowFilter="filter"`	The additional filter for searches for request workflow instances. [In; optional; accepts a JSTL expression.]

8.2.23. GetActionInfo Tag

8.2.23.1. Description:

The convenience tag sets some scoped variables to facilitate access to information about the currently processed Struts action, like the form, path and forward names.

The tag attributes include:

- The variable names for:
- The action path like “/getUsers“.
- The extended action path like “/getUsers.do“.
- The action form.
- The action form name.
- The action form property map.

- The action forwards, a map mapping each action forward name to the value Boolean.TRUE.
- The variable scope. All variables are created in the same scope

8.2.23.2. Usage:

```
<ctrl:getActionInfo Attributes/>
```

8.2.23.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`varExtendedPath="varName"`	The name of the variable to accept the action path (String). [Out; optional; default: “extendedActionPath”; accepts a JSTL expression.]
`varForm="varName"`	The name of the variable to accept the action form (DynaActionForm). [Out; optional; default: “form”; accepts a JSTL expression.]
`varFormMap="varName"`	The name of the variable to accept the action form property map (java.util.Map). [Out; optional; default: “formMap”; accepts a JSTL expression.]
`varFormName="varName"`	The name of the variable to accept the action form name (String). [Out; optional; default: “formName”; accepts a JSTL expression.]
`varForwards="varName"`	The name of the variable to accept the action forwards (java.util.Map). [Out; optional; default: “actionForwards”; accepts a JSTL expression.]
`varPath="varName"`	The name of the variable to accept the action path (String). [Out; optional; default: “actionPath”; accepts a JSTL expression.]

8.2.24. GetLoginUser Tag

8.2.24.1. Description:

The tag evaluates a user's input into a login form, searches for a matching user object in the directory and returns its DN. Note that the user is not authenticated; any password entered into the login form is ignored.

The tag checks whether the user has entered the required minimum number of characters into the login form. If not, the search is skipped and a DN is not returned. The minimum number of characters is defined by the application initialization parameter "com.siemens.webMgr.login.minChars".

The tag attributes include:

- The property map of the login form the user has submitted.
- The object class and the search base for user objects.

  The tag returns the user's DN if exactly one object matches the data entered into the login form, provided the user has entered the minimum number of characters into the login form. Otherwise, the result variable is removed from its scope.

8.2.24.2. Usage:

`<ctrl:getLoginUser` *`Attributes`*`/>`

8.2.24.3. Attributes:

`attributes="`*`attributes`*`"`	The property map of the login form (java.util.Map). [In; required; accepts a JSTL expression.]
`objectclass="`*`objectclass`*`"`	The object class for user objects (String). [In; optional; default: "*"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "session"; accepts a JSTL expression.]
`searchBase="`*`dn`*`"`	The search base (String). [In; optional; default: the value of the application initialization parameter "com.siemens.webMgr.ldap.baseDN"; accepts a JSTL expression.]

`var="varName"`	The name of the variable to accept the user's DN (String). [Out; optional; default: "com.siemens.webMgr.loginDN"; accepts a JSTL expression.]

8.2.25. GetMemberInfo Tag

8.2.25.1. Description:

The tag returns DN and type of a given group member. The member must be specified by the group's DN and the member's identifier within that group.

There are two group member types, "account" and "user". An account member identifier is usually not the valid DN of an entry in the database but an artificial string whose exact format depends on the type of group. A user member identifier is usually the user's DN.

The result variables are removed from their scope in case member DN and type cannot be determined.

The tag throws a JSP exception in case of errors,

8.2.25.2. Usage:

```
<ctrl:getMemberInfo Attributes/>
```

8.2.25.3. Attributes:

`groupId="dn"`	The group DN (String). [In; required; accepts a JSTL expression.]
`memberId="memberId"`	The member id, as obtained from a group member list (String). [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: "page"; accepts a JSTL expression.]
`varDN="varName"`	The name of the variable to accept the member DN (String). [Out; optional; default: "dn"; accepts a JSTL expression.]

`varType="`*`varName`*`"`	The name of the variable to accept the member type (String). [Out; optional; default: “type“; accepts a JSTL expression.]

8.2.26. GetObjectType Tag

8.2.26.1. Description:

The tag returns the object type of a given object.

8.2.26.2. Usage:

```
<ctrl:getObjectType Attributes/>
```

8.2.26.3. Attributes:

`objectId="`*`dn`*`"`	The object’s DN (String). [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the object type (String]). [Out; optional; default: “objectType“; accepts a JSTL expression.]
`varCap="`*`varName`*`"`	The name of the variable to accept the object type with upper-case initial letter (String). [Out; optional; default: “objectTypeCap“; accepts a JSTL expression.]

8.2.27. GetRoleParameters Tag

8.2.27.1. Description:

The tag returns the parameters that apply to assignments of a given role. For each parameter, the tag returns its name, role name, type and, if applicable, proposals or root nodes. The parameter values are left unspecified.

The return variable is removed from its scope in case no parameters apply.

The tag throws a JSP exception in case of errors,

8.2.27.2. Usage:

```
<ctrl:getRoleParameters Attributes/>
```

8.2.27.3. Attributes:

`objectId="dn"`	The role DN (String). [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the role parameters (Parameter[]). [Out; optional; default: “roleParameters“; accepts a JSTL expression.]

8.2.28. GetStartAction Tag

8.2.28.1. Description:

The tag returns the start action for the logged in user, as defined in the respective Tiles definition in **menu-defs.xml**.

If no start action applies to the current user the result variable is removed from its scope.

The tag throws a JSP exception in case of errors.

8.2.28.2. Usage:

```
<ctrl:getStartAction Attributes/>
```

8.2.28.3. Attributes:

`definition="definition"`	The name of the tiles definition for the start actions. [In; optional; default: “.startActions“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

`var="varName"`	The name of the variable to accept the start action (Parameter[]). [Out; optional; default: “startAction“; accepts a JSTL expression.]

8.2.29. InstanceOf Tag

8.2.29.1. Description:

The tag checks if a Java object is an instance of a given Java class or interface.

8.2.29.2. Usage:

```
<ctrl:instanceof Attributes/>
```

8.2.29.3. Attributes:

`object="object"`	The Java object. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`type="type"`	The name of the Java class or interface. [In; required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the result (Boolean). [Out; optional; default: “result“; accepts a JSTL expression.]

8.2.29.4. Body Content:

The tag accepts any body content. The content is evaluated only if the object is an instance of the class or interface.

8.2.30. IsActionForward Tag

8.2.30.1. Description:

The checks whether a given name is a valid forward name of the current Struts action.

8.2.30.2. Usage:

```
<ctrl:isActionForward Attributes/>
```

8.2.30.3. Attributes:

`name="`*`name`*`"`	The name to check. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`toLowerCase="true"|"false"`	Whether to convert the name to lower cases before checking. [In; optional; default:”false”; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the check result (Boolean). [Out; optional; default “isLoggedIn”; accepts a JSTL expression.]

8.2.30.4. Body Content:

The tag accepts any body content. The content is evaluated only if the given name is a valid forward name.

8.2.31. IsLoggedIn Tag

8.2.31.1. Description:

The checks whether the user has already successfully logged in.

The tag attributes include a flag to do the reverse check.

In case the flag is true, the tag returns:

- Boolean.TRUE, if the user is logged in.
- Boolean.FALSE, if the user is not logged in.

In case the flag is false, the tag returns:

- Boolean.FALSE, if the user is logged in.
- Boolean.TRUE, if the user is not logged in.

8.2.31.2. Usage:

<ctrl:isLoggedIn *Attributes*/>

8.2.31.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`value="true"|"false"`	Whether to check if the user is logged in (true), or not logged in (false). [In; optional; default “true”; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the check result (Boolean). [Out; optional; default “isLoggedIn”; accepts a JSTL expression.]

8.2.31.4. Body Content:

The tag accepts any body content. The content is evaluated only if the user is logged in.

8.2.32. IsSSOClient Tag

8.2.32.1. Description:

The tag checks whether the user has logged in via a single sign-on mechanism.

The tag attributes include a flag to do the reverse check.

If the flag is true, the tag returns:

- Boolean.TRUE, if the user is logged in.
- Boolean.FALSE, if the user is not logged in.

If the flag is false, the tag returns:

- Boolean.FALSE, if the user is logged in.
- Boolean.TRUE, if the user is not logged in.

8.2.32.2. Usage:

`<ctrl:isSSOClient` *`Attributes`*`/>`

8.2.32.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

`value="true"|"false"`	Whether to check if the user is logged via SSO (true), or not logged in via SSO (false). [In; optional; default “true”; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the check result (boolean). [Out; optional; default “isSSOClient”; accepts a JSTL expression.]

8.2.32.4. Body Content:

The tag accepts any body content. The content is evaluated only if the user has logged in via single sign-on.

8.2.33. ListEntries Tag

8.2.33.1. Description:

The tag fills a table property of a form bean with attributes from a search result. The property must be one of the objectSets supported by the Web-API method “listEntries”.

The search context is used to specify

- The index of the last entry to be returned (The index of the first element in the search result is 1).
- The index of the first entry to be returned (only for property “workItems”; must be 1 for all other properties).
- Criteria for sorting the search result (only for property “workItems”).

The search context returns the total number of objects found.

8.2.33.2. Usage:

`<ctrl:listEntries` *`Attributes`*`/>`

8.2.33.3. Attributes:

`context="`*`context`*`"`	The search context (of type SearchContext). [In; optional; accepts a JSTL expression.]
`form="`*`form`*`"`	The form containing the property to be populated. [In; required; accepts a JSTL expression.]

`objectId="dn"`	The DN of the master object for a search operation. [In; optional; accepts a JSTL expression.]
`property="property"`	The name of the form property to be populated. The property must be of type DirectoryEntryBean[]. [In; required; accepts a JSTL expression.]

8.2.34. ListReports Tag

8.2.34.1. Description:

The tag retrieves a list of reports and assigns it to a form property.

The tag attributes include:

- The filter for the report search.
- The search base for the objects to be handled by the report.

The tag returns

- The string “ok” on success.
- The string “objectNotFound” in case no report was found.
- The string “failed” in case of a failure.

8.2.34.2. Usage:

```
<ctrl:listReports Attributes/>
```

8.2.34.3. Attributes:

`filter="filter"`	The filter for the report search. [In; required; accepts a JSTL expression.]
`formName="formName"`	The name of the form to accept the report list. [In; required; accepts a JSTL expression.]
`objectId="dn"`	The DN of the search base for the entries to be handled by the report. [In; required; accepts a JSTL expression.]
`propertyName="propertyName"`	The name of the form property to accept the report list. [In; required; accepts a JSTL expression.]

`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the result. [Out; optional; default: “result”; accepts a JSTL expression.]

8.2.35. Log Tag

8.2.35.1. Description:

The tag writes a log message or detailed information about the current HTTP request or session to the log file.

8.2.35.2. Usage:

`<ctrl:log` *`Attributes`*`/>`

8.2.35.3. Attributes:

`level="error"|"info"|"debug"`	The log level. [In; optional; default: “debug“; accepts a JSTL expression.]
`message="`*`message`*`"`	The log message. [In; optional; accepts a JSTL expression.]
`request="true"|"false"`	Whether to log detailed information about the HTTP request (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]
`session="true"|"false"`	Whether to log detailed information about the HTTP session (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]

8.2.36. Login Tag

8.2.36.1. Description:

The tag authenticates a user against the provisioning directory.

The tag attributes include:

- Host name and port number of the LDAP server of the provisioning database, as well as a flag indicating whether to connect to the server via SSL. By default, this information is taken from the corresponding application initialization parameters.
- The user credentials: DN and password. If left unspecified, these values are searched for in
- The requests parameters (“loginDN” and “password”).
- Externally provided credentials (in the case of single sign-on).
- Session-scoped variables (“com.siemens.webMgr.loginDN” and “com.siemens.webMgr.password”).
- A flag indicating whether login attempts by the special user ANYONE are accepted or rejected.
- A flag indicating whether to check the user’s password status flags after successful authentication.

  The tag returns the string:

- OK: The user has been successfully authenticated.
- TECH: The technical bind was successful, but the user could not be authenticated.
- NONE: Neither the user authentication nor the technical bind was successful.
- ANYONE_REJECTED: A login attempt of ANYONE was rejected.

If the password flags are to be checked after successful login, the following page-scoped variables are set:

- pwdStatus:
- ok: The password is OK.
- expired: The password has expired.
- reset: The password must be reset.
- warning: An expiration warning should be displayed.
- pwdDays. For an expiration warning, the number of days before the password expires.

The following session-scoped attributes can be modified or removed:

- com.siemens.webMgr.loginDN
- com.siemens.webMgr.password
- com.siemens.webMgr.userData

8.2.36.2. Usage:

```
<ctrl:login Attributes/>
```

8.2.36.3. Attributes:

`allowAnyone="true"|"false"`	Whether the user ANYONE is accepted or rejected. [In; optional; default: “false“; accepts a JSTL expression.]
`checkPwdStatus="true"|"false"`	Whether to check the user’s password status flags after authentication. [In; optional; default: “false“; accepts a JSTL expression.]
`dn="`*`dn`*`"`	The DN of the user to be authenticated. [In; optional; default: “page“; accepts a JSTL expression.]
`Password="`*`password`*`"`	The password of the user to be authenticated. [In; optional; default: “page“; accepts a JSTL expression.]
`port="`*`port`*`"`	The port of the directory server. [In; optional; default: the value of the application initialization parameter “com.siemens.webMgr.ldap.port”; accepts a JSTL expression.]
`server="`*`server`*`"`	The host name or IP address of the directory server. [In; optional; default: the value of the application initialization parameter “com.siemens.webMgr.ldap.host”; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`ssl="true"|"false"`	Whether to connect to the server via SSL. [In; optional; default: the value of the application initialization parameter “com.siemens.webMgr.ldap.ssl”; accepts a JSTL expression.]
`skipEffectiveAuth="true"|"false"`	Whether to skip checking the authentication of the effective user. To be used in case of single sign-on where the user password is unknown. [In; optional; default: “false“; accepts a JSTL expression.]

`var="varName"`	The name of the variable to accept the operation's result. [Out; optional; accepts a JSTL expression.]

8.2.37. Logout Tag

8.2.37.1. Description:

The tag cleans up the resources associated with a user's session. The connection to the directory is shut down. Internal caches are destroyed. Session-scoped variables are removed.

8.2.37.2. Usage:

```
<ctrl:logout/>
```

8.2.37.3. Attributes:

None.

8.2.38. ModifyDelegation Tag

This tag applies to the new delegation implementation only.

8.2.38.1. Description:

The tag updates an existing delegation.

The tag attributes include:

- The DN of the delegation to be updated.
- The form bean to get the delegation's updated attributes from.
- The list of the names of the attributes to be copied from the form bean to the updated delegation. Attributes "dxrAssignFrom", "dxrAssignTo" and "dxrOperationImp" are ignored.
- Whether to ignore warnings about delegation cycles caused by the updated delegation.

The tag returns

- A string that indicates the operation's result:
- ok: The operation was successful.
- delegationsDisabled: The new delegations are not enabled.
- circularDelegationChain: The operation failed since the updated delegation would lead to a circular delegation chain.
- invalidSubstitute: The new substitute coincides with the delegator.
- dateConflict: The updated delegation's start and end date are inconsistent.

- failed: The operation failed for another reason.

8.2.38.2. Usage:

```
<ctrl:modifyDelegation Attributes/>
```

8.2.38.3. Attributes:

`attributes="attributes"`	The list of additional attribute names for the delegation. [In; optional; accepts a JSTL expression.]
`ignoreWarnings="true"|"false"`	Whether to ignore circular delegation warnings (true), or not (false). [In; optional; default: “false”; accepts a JSTL expression.]
`objectId="dn"`	The DN of the delegation. [In; required; accepts a JSTL expression.]
`operation="approve"|"grant"|`	The delegation operation. [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`source="form"`	The form being the source for the delegation attribute updates (DynaActionForm) [In; required; accepts a JSTL expression.]
`substitute="dn"`	The DN of the substitute. [In; required; accepts a JSTL expression.]

8.2.39. Move Tag

8.2.39.1. Description:

The tag moves one or more objects to a new parent, access rights permitting.

The tag attributes for moving a single object include:

- The DN of the object to move.

The tag returns

- The string “ok” in case of success. In this case, the tag also returns the object’s new DN.
- The string “disallowed” in case of insufficient access rights.
- The string “failed” in case of error.

The tag attributes for moving multiple objects include:

- The DNs of the objects to move.

The tag returns

- The string “ok” in case of success.
- The string “failed” in case of errors of insufficient access rights. In this case, the tag also returns which objects have been moved, which objects could not be moved due to insufficient access rights and which ones failed to be moved.

If no object is specified the tag returns “ok”.

8.2.39.2. Usage:

```
<ctrl:delete Attributes/>
```

8.2.39.3. Attributes:

`varNewDN="varName"`	In case a single object has been successfully moved only: The name of the variable to accept the moved object’s new DN (String). [Out; optional; default: “newDN“; accepts a JSTL expression.]
`objectId="dn"`	The DN of the object to be moved (String). [In; optional; accepts a JSTL expression.]
`objectIds="dns"`	The DNs of the objects to be moved (String[]) [In; optional; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation’s result (String). [Out; optional; default: “result“; accepts a JSTL expression.]

`scope="page"|"request"|"session"|"application"`	The scope for the result variables. [In; optional; default: “page“; accepts a JSTL expression.]
`varMoved="`*`varName`*`"`	In case a request for multiple moves failed only: The name of the variable to accept the DNs of the entries that have been moved (String[]). [Out; optional; default: “moved“; accepts a JSTL expression.]
`varDisallowed="`*`varName`*`"`	In case a request for multiple moves failed only: The name of the variable to accept the DNs of the entries that could not be moved due to insufficient access rights (String[]). [Out; optional; default: “disallowed“; accepts a JSTL expression.]
`varFailed="`*`varName`*`"`	In case a request for multiple moves failed only: The name of the variable to accept the DNs of the entries that could not be moved due to some error (String[]). [Out; optional; default: “failed“; accepts a JSTL expression.]

8.2.40. Param Tag

8.2.40.1. Description:

The tag sets a parameter of an enclosing setMessage tag, optionally replacing “\n” substrings (2 characters, a backslash, followed by the letter n) in the value by new line characters (‘\n’).

The tag throws a JSP exception if not enclosed by a setMessage tag.

8.2.40.2. Usage:

`<ctrl:param` *`Attributes`*`/>`

8.2.40.3. Attributes:

`replaceNewLine="true"|"false"`	Whether to replace "\n" with a new line character when inserting the parameter value into the message. [In; optional; default: “false“; accepts a JSTL expression.]
`value="`*`value`*`"`	The message parameter value. [In; optional; default: the empty string; accepts a JSTL expression.]

8.2.40.4. Ancestor Tag:

`setMessage`	A setMessage tag

8.2.41. PurgeSession Tag

8.2.41.1. Description:

The tag removes session-scoped variables in order to keep the size of user sessions at a reasonable level. In the default Web Center applications, the tag is invoked by a servlet filter on specific requests.

The tag attributes include:

- An array of regular expressions for variable names.
- An array of class names.

A variable is removed if

- Its name matches one of the regular expressions for variable names.
- The class (or any base class or interface) of the object it refers to equals an item in the class name array.

8.2.41.2. Usage:

`<ctrl:purgeSession` *`Attributes`*`/>`

8.2.41.3. Attributes:

`attrNames="`*`attrNames`*`"`	An array of regular expressions for names of session variables to remove. [In; optional; accepts a JSTL expression.]
`classNames="`*`classNames`*`"`	An array of the class names of session variables to remove. [In; optional; accepts a JSTL expression.]

`keepMessage="true"|"false"`	Whether to retain message related session variables. [In; optional; default: “false“; accepts a JSTL expression.]

8.2.42. RDN Tag

8.2.42.1. Description:

The tag provides some utility methods to handle relative distinguished names.

The tag attributes include:

- The RDN argument for the method, or type and value for concatenating an RDN.
- The method name.

The method getRDN returns the RDN concatenated from a given type and an unescaped value.

The method getType returns the attribute type of an RDN.

The method getValue returns the unescaped attribute value of an RDN.

8.2.42.2. Usage:

`<ctrl:rdn` *`Attributes`*`/>`

8.2.42.3. Attributes:

`method="getRDN"|"getType"|"getValue"`	The method to invoke (String). [In; optional; default: getRDN; accepts a JSTL expression.]
`rdn="`*`rdn`*`"`	The first DN argument for the method (String). [In; optional/required depending on method; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation’s result (String or Boolean). [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

`type="type"`	The RDN type (String) [In; optional/required depending on method; accepts a JSTL expression.]
`value="value"`	The RDN value (String) [In; optional/required depending on method; accepts a JSTL expression.]

8.2.43. ReadPassword Tag

8.2.43.1. Description:

The tag returns the clear text password of an object (i.e. the decrypted dxmPassword attribute), access rights permitting.

In case of success, the variable result is set to “ok” while the clear text password is returned in variable password.

The variable result is set to “disallowed” in case of insufficient access rights, and to “failed” in case of errors, while the password variable is removed from its scope.

8.2.43.2. Usage:

`<ctrl:readPassword Attributes/>`

8.2.43.3. Attributes:

`objectId="dn"`	The DN of the object whose password is to be returned (String). [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the result (String). [Out; optional; default: “result; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`varPassword="varName"`	The name of the variable to accept the password String. [Out; optional; default: “password; accepts a JSTL expression.]

8.2.44. ReleaseUserLocks Tag

8.2.44.1. Description:

The tag releases a user's locks for logging in with name and password and/or for authenticating via challenge response. It resets all attributes related to the locks.

The tag attributes include:

- The user's DN.
- Which locks to release: "password", "challengeResponse" or "password, challengeResponse". If left unspecified, both locks are released.

The tag returns a string that indicates the operation's result:

- ok: The locks have been released.
- disallowed: The logged-in user is not allowed to perform the operation.
- failed: The locks could not be released.

8.2.44.2. Usage:

`<ctrl:releaseUserLocks Attributes/>`

8.2.44.3. Attributes:

`locks="locks"`	The locks to be released (String). [In; optional; accepts a JSTL expression.]
`objectId="dn"`	The DN of the object whose locks are to be released (String). [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the result (String). [Out; optional; default: "result; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2.45. RemoveToken Tag

8.2.45.1. Description:

The tag removes the CSRF token from the session.

8.2.45.2. Usage:

```
<ctrl:removeToken/>
```

8.2.46. RemoveVar Tag

8.2.46.1. Description:

The tag removes a variable from its scope.

8.2.46.2. Usage:

```
<ctrl:removeVar Attributes/>
```

8.2.46.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The variable scope. [In; optional; default: “page“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to remove. [Out; required; accepts a JSTL expression.]

8.2.47. Report Tag

8.2.47.1. Description:

The tag searches for reports, generates a selected report or returns a link to download a generated report.

If attribute reportFile is not empty, the tag returns a link to download the most recently generated report. The link can be used to display the report in a frame. Note that there is at most one generated report per user session. The tag attributes include:

- The form and the name of the form property to accept the link (both required).

No result is returned. The result variable is removed from its scope.

If attribute selectedReportIndex is a valid index, the tag generates the report with the specified index in the list of reports assigned to the provided form. The generated report is temporarily stored in a file below the application’s temporary working directory. The tag attributes include:

- The form with the report list (required).
- The form property name of the report list (required).
- The index of the selected report (required).
- The form property names of the base object (required) and the scope for the report.
- A list of accepted report types, e.g. “HTML,XML,TXT”. The generated report is of the first

type in the list supported by the report.

- The prefix for report file names.

The result variable is set to the type of the generated report.

Otherwise, the tag searches for reports and populates the report form with the resulting list. The tag attributes include:

- The form to accept the report list (required).
- The form property name of the report list (required).
- The form property names of the master object (required for the report list.
- The prefix for report file names.

No result is returned. The result variable is removed from its scope.

8.2.47.2. Usage:

`<ctrl:report *Attributes*/>`

8.2.47.3. Attributes:

`filter="*filter*"`	The form property name of the filter for searching for reports, or the filter for report generation. [In; optional; accepts a JSTL expression.]
`formObject="*form*"`	The form for listing reports, or for displaying a generated report (DynaActionForm). [In/Out; required; accepts a JSTL expression.]
`listAttribute="*propName*"`	The name of the form property of the report list, or to accept the report list. [In/Out; optional/required; accepts a JSTL expression.]
`namePrefix="*namePrefix*"`	The prefix for report file names. [In; optional; default: “Report_“; accepts a JSTL expression.]
`objectId="*propName*"`	The form property name of the master object for searching for reports, or the base object for report generation. [In; optional/required; accepts a JSTL expression.]

`reportFile="propName"`	The name of the form property to accept the download link (String). [Out; optional/required; accepts a JSTL expression.]
`reportFilter="propName"`	The form property name of the filter for report generation. [In; optional; accepts a JSTL expression.]
`reportScope="basenode"|"onelevel"|"subtree"|"`	The form property name of the scope for report generation. [In; optional; default: "basenode"; accepts a JSTL expression.]
`reportTypes="types"`	A comma-separated list of report types accepted by the caller. [In; optional; default: “HTML“; accepts a JSTL expression.]
`resultScope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`selectedReport`	The index of the selected report. [In; optional/required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the report type. [Out; optional; accepts a JSTL expression.]

8.2.48. ResetLoginStatus Tag

8.2.48.1. Description:

The tag resets the login status of the currently active user by deleting the values of the attributes dxrPwdAccountLockedTime and dxrPwdFailureTime.

8.2.48.2. Usage:

```
<ctrl:resetLoginStatus Attributes/>
```

8.2.48.3. Attributes:

`result="varName"`	The name of the variable to accept the operation's result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]

8.2.49. ResetObject Tag

8.2.49.1. Description:

The tag discards any unsaved changes to an object.

8.2.49.2. Usage:

```
<ctrl:resetObject Attributes/>
```

8.2.49.3. Attributes:

`objectId="dn"`	The DN of object to reset. If left unspecified, the tag does nothing. [In; optional; accepts a JSTL expression.]

8.2.50. Select Tag

8.2.50.1. Description:

The tag assigns the DN of a selected object or the value of a selected object property to a scoped variable.

A selected object is identified by its row index in an array of DirectoryEntryBeans. Index "0" selects the first item.

A selected object property is identified by the row index of the object, and by the column index of the property in an array of DirectoryEntryBeans. The row;column pair "0;0" selects the first property of the first item.

Selecting an entry via column index can be suppressed by setting attribute selectProperty to "false". In this case, always the object displayed in the row is selected. For example, if a user list displays the users' managers, clicking on a manager will select the user, not the manager.

8.2.50.2. Usage:

```
<ctrl:select Attributes/>
```

8.2.50.3. Attributes:

`dataset="`*`dataset`*`"`	The item list (DirectoryEntryBean[]). [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`selection="`*`selection`*`"`	The index or row and column of the selected entry. [In; optional; accepts a JSTL expression.]
`selectProperty="true"|"false"`	Whether to select objects referenced by a property of a list item (true), or whether to select always the list item (false). [In; optional; default: "true"; accepts a JSTL expression.]
`selectSingleItem="true"|"false"`	Whether to auto-select the first item if the list has size 1. [In; optional; default: "false"; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the result. [Out; optional; accepts a JSTL expression.]

8.2.51. Selection Tag

8.2.51.1. Description:

The tag either

- Removes the selected entries from a form property representing a table. Note that only the form property is updated; the entries are not deleted from the database.
- Returns a map providing easy access to the set of selected entries in a table. The map is used to perform an operation on the selected entries one by one. It contains the following entries:
- form: The form (DynaActionForm).
- formField: The form property name (String).
- forward: The provided forward action name (String).
- index: The position of the currently handled entry; initially 0 (Integer).
- size: Number of selected entries (Integer).
- dns: The DNs of the selected entries (String[]).

- entries: The selected entries (DirectoryEntryBean[]).

The selected entries can be either specified via their indexes in table or via their DNs.

8.2.51.2. Usage:

`<ctrl:selection` *`Attributes`*`/>`

8.2.51.3. Attributes:

`form="`*`form`*`"`	The form containing the table (DynaActionForm). [In; optional; accepts a JSTL expression.]
`formFieldName="`*`formFieldName`*`"`	The form field name of the table (String). [In; optional; accepts a JSTL expression.]
`forward="`*`forward`*`"`	The context menu forward action name (String). [In; optional; accepts a JSTL expression.]
`operation="get"|"remove"`	The operation type. [In; optional; default: "remove"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for varSelection. [In; optional; default: “session“; accepts a JSTL expression.]
`selectedDNs="`*`selectedDNs`*`"`	The DNs of the selected table entries (String[]). [In; optional; accepts a JSTL expression.]
`selectedIndexes="`*`selectedIndexes`*`"`	The indexes of the selected table entries (int[]). Takes precedence over selectedDNs. [In; optional; accepts a JSTL expression.]
`selection="`*`selection`*`"`	The selection obtained from a prior call to tag getSelection. For operation type remove only. [In; optional; accepts a JSTL expression.]
`varSelection="`*`varName`*`"`	The name of the variable to accept the result. [Out; optional; default: “com.siemens.webMgr.selection”; accepts a JSTL expression.]

8.2.52. Set Tag

8.2.52.1. Description:

The tag sets or updates attributes of an entry (in memory), saves unsaved updates to the directory, or resets all unsaved updates.

The tag attributes include:

- The DN of the object to be updated or reset.
- A comma-separated list of the names of attributes to be copied from the source to the object. Each attribute whose name is prefixed with an exclamation mark ("!") is ignored.
- The source for the new attribute values, which can be:
- A form bean (DynaActionForm): If the attribute list is empty, all non-read-only form properties are copied. Otherwise, only the specified properties are copied.
- An array (Object[]): If the attribute list contains just one attribute name, the value of that attribute is set to the array. Otherwise, the array item at index i is assigned to the attribute at index i.
- Any other Object: The object is assigned to the first attribute in the list.
- The operation type, which can be:
- reset: Reset all unsaved updates to the object; ignore any updates to the source. The result is “unchanged” or “reset”.
- storeOnly: Copy updates from the source to the object, but don’t save the object to the directory. The result is always “saved”.
- saveOnly: Save the object to the directory. Ignore any new changes to the source. Don’t save in case of warnings like SoD violations. The result is “unchanged”, “saved” or “warning”.
- ignoreWarnings: Save the object to the directory. Ignore any new changes to the source. Save without further confirmation in case of warnings like SoD violations. The result is “unchanged” or “saved”.
- save: Copy updates from the source to the object. Save the object to the directory. Don’t save in case of warnings like SoD violations. The result is “unchanged”, “saved” or “warning”.
- saveNoFrills: Copy updates from the source to the object. Then just save the object to the directory but don’t perform any special tasks like SoD violation checks or role resolution. The result is “unchanged” or “saved”.
- A flag that indicates whether only assignments should be saved. Assigning privileges to an object doesn’t require the access right to modify the object. Other attribute changes (possibly inadvertently or automatically performed) usually will require it and may therefore prohibit assignments.
- A flag that controls operation signing. The flag is evaluated only if the object is to be saved.
- init: Don’t save the object. Just generate and return an XML document to sign the changes.

- finish: Validate the provided signed document. Save the object to the directory, and save the signed document.
- The operation due date.

8.2.52.2. Usage:

`<ctrl:set Attributes/>`

8.2.52.3. Attributes:

`assignmentsOnly="true"|"false"`	Whether to save assignments only (true) or all attributes (false). [In; optional; default: "false"; accepts a JSTL expression.]
`attributes="attributes"`	The attribute set to be stored. [In; optional; accepts a JSTL expression.]
`dueDate="dueDate"`	The operation due date. [In; optional; accepts a JSTL expression.]
`finally="ignoreWarnings"|"reset"|"save"|"saveNoFrills"|"saveOnly"|"storeOnly"`	The operation type. [In; optional; default: “save”; accepts a JSTL expression.]
`objectId="dn"`	The DN of the entry to be updated. [In; optional; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the result (String). [Out; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for result and newDN. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeTextToBeSigned="page"|"request"|"session"|"application"`	The scope for varTextToBeSigned. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeWarning="page"|"request"|"session"|"application"`	The scope for varWarning. [In; optional; default: “request“; accepts a JSTL expression.]
`signedData="signedData"`	The signed XML document. [In; optional; accepts a JSTL expression.]

`signState="init"|"finish"`	Whether to start or finish signing the operation. [In; optional; accepts a JSTL expression.]
`source="`*`source`*`"`	The source of the data to be stored. [In; required; accepts a JSTL expression.]
`varNewDN="`*`varName`*`"`	The name of the variable to accept the object's possibly changed DN after changes to the object (String). [Out; optional; default: "newDN"; accepts a JSTL expression.]
`varTextToBeSigned="`*`varName`*`"`	The name of the variable to accept the XML document to signed (String). [Out; optional; accepts a JSTL expression.]
`varWarning="`*`varName`*`"`	The name of the variable to accept the message in case of warnings (String). [Out; optional; accepts a JSTL expression.]

8.2.53. SetActionForward Tag

8.2.53.1. Description:

A convenience tag to set the action forward in an action page. If the specified forward name is not configured for the current action, the supplied default forward name is set (without further checks).

The attribute returnCode let's you define the tag's return code which defines how processing of the JSP page continues. With skipPage, you can cause processing to stop.

8.2.53.2. Usage:

`<ctrl:setActionForward` *`Attributes`*`/>`

8.2.53.3. Attributes:

`default="`*`forwardName`*`"`	The default forward name. [In; optional; accepts a JSTL expression.]
`name="`*`forwardName`*`"`	The forward name. [In; optional; accepts a JSTL expression.]
`returnCode="evalPage"|"skipPage"|"evalBodyInclude"|"skipBody"`	The tags return code. [In; optional; default: "evalPage"; accepts a JSTL expression.]

`toLowerCase="true"|"false"`	Whether to convert the provided forward name to lower case. [In; optional; default: "false"; accepts a JSTL expression.]

8.2.54. SetBinary Tag

8.2.54.1. Description:

The tag sets or updates a binary attribute of an entry. The changes are not saved to the database,

The tag attributes include:

- The DN of the object to be updated.
- The name of the binary attribute.
- The values to be added (DirFileParam[]), as obtained from invoking the DirX JSP tag setParams.
- A list of expected content types that the items in the DirFileParam[] must match (case ignore string match or regular expression match).
- Whether to check the validity of the provided values; this is implemented for X.509 certificate attributes only.
- The values to be deleted (byte[][] or String[]):
- byte[]: The binary value to be deleted.
- String: The MD5-SHA1 hash of the value to be deleted.

The tag returns

- The string “ok” in case the property has been updated.
- The string “unchanged” in case no property updates were necessary and done.
- The string “invalid” in case the type of a value to be added doesn’t match any of the expected types.

8.2.54.2. Usage:

`<ctrl:setBinary` *`Attributes`*`/>`

8.2.54.3. Attributes:

`add="`*`addValues`*`"`	The values to be added (DirFileParam[]). [In; optional; accepts a JSTL expression.]
`attributes="`*`attributes`*`"`	The name of the binary property. [In; required; accepts a JSTL expression.]

`checkContent="checkContent"`	Whether to perform content checks. [In; optional; default:"true"; accepts a JSTL expression.]
`contentTypes="contentTypes"`	A comma separated list of expected content types. [In; optional; accepts a JSTL expression.]
`delete="deleteValues"`	The values to be deleted (byte[][]), or their MD5-SHA1-hashes (String[]). [In; optional; accepts a JSTL expression.]
`objectId="dn"`	The DN of the entry to be updated. [In; required; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the result (String). [Out; optional; default: "result"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for result and varInvalidFile. [In; optional; default: "page"; accepts a JSTL expression.]
`varInvalidFile="varName"`	The name of the variable to accept the name of a file whose type is invalid (String). [Out; optional; default: "invalidFile"; accepts a JSTL expression.]

8.2.55. SetLanguage Tag

8.2.55.1. Description:

The tag changes the language of the current user.

If no new language is provided or the provided language is not supported, the tag takes no action.

8.2.55.2. Usage:

```
<ctrl:setLanguage Attributes/>
```

8.2.55.3. Attributes:

`value="locale"`	The current user's new language. [In; optional; accepts a JSTL expression.]

8.2.56. SetLocale Tag

8.2.56.1. Description:

The tag assigns the locale of the currently logged-in user to the request-scoped variable ”javax.servlet.jsp.jstl.fmt.locale.request”.

The variable is evaluated by the internationalization tags of the JSP standard tag library (like <fmt:setBundle>) and must therefore be set before invoking any such tag.

8.2.56.2. Usage:

```
<ctrl:setLocale Attributes/>
```

8.2.57. SetMessage Tag

8.2.57.1. Description:

A convenience tag to set message-related session variables. A message has a key, a confirmation flag, and parameters.

The key is the reference to the message text in a resource bundle (which is the file **text.properties** for the default Web Center applications).

The confirmation flag indicates whether the end user must confirm the message. A confirmation message box has two buttons, one to confirm and another one to cancel the operation.

A message text may include placeholders ({0}, {1} etc.) which are replaced with message parameters at runtime. Message parameters usually contain more detailed information about a problem. Parameters may be supplied as an array of Strings, or as any other object. In the latter case, the object is converted into a String using its toString method.

The insert mode defines how the parameters are combined with previously stored ones. The new ones can be appended or prepended to the existing ones, or they can just replace them.

If the message key is left unspecified, the current message-related variables are removed from the session scope.

Messages confirming the successful outcome of an operation should be labeled with success=”true”. Display of these messages is controlled by a parameter in file **webCenter.properties**.

8.2.57.2. Usage:

```
<ctrl:setMessage Attributes>
Body content
</ctrl:setMessage>
```

8.2.57.3. Attributes:

`clearError="true"|"false"`	Whether to clear the session's error code and error cause. [In; optional; default: "false"; accepts a JSTL expression.]
`confirm="true"|"false"`	The confirmation flag. [In; optional; default: "false"; accepts a JSTL expression.]
`defaultKey="`*`messageKey`*`"`	The default message key; evaluated in case no message with the key specified by attribute "key" was found. [In; optional; accepts a JSTL expression.]
`insertMode="append"|"override"|"prepend"`	How to mix the message parameters with existing ones. [In; optional; default: "override"; accepts a JSTL expression.]
`key="`*`messageKey`*`"`	The message key. [In; optional; accepts a JSTL expression.]
`param="`*`messageParameter`*`"`	Message parameters (String[] or Object). [In; optional; accepts a JSTL expression.]
`success="true"|"false"`	A flag indicating whether the message confirms the successful outcome of an operation. [In; optional; default: "false"; accepts a JSTL expression.]

8.2.57.4. Body Content:

The tag accepts any JSP content in its body, especially the following child tag:

`param`	A message parameter.

8.2.58. SetOption Tag

The tag has been withdrawn for security reasons. Use the tags **SetStyle** and **SetLanguage** instead.

8.2.59. SetRoleParameters Tag

8.2.59.1. Description:

The tag sets the parameter values for a role assignment. The values must be provided in a map, like an HTTP request parameter value map.

For each role parameter, if the map contains a value with the parameter name as key, that value is assigned to the parameter; otherwise, any existing parameter value is removed.

8.2.59.2. Usage:

```
<ctrl:setRoleParameters Attributes/>
```

8.2.59.3. Attributes:

`target="target"`	The role parameter array. [In; optional; accepts a JSTL expression.]
`valueMap="valueMap"`	The role parameter value map (java.util.Map). [In; optional; accepts a JSTL expression.]

8.2.60. SetStyle Tag

8.2.60.1. Description:

The tag changes the style (font size) of the current user.

If no new style is provided or the provided style is not supported, the tag sets the style to the default one defined in file **webCenter.properties**, or as a last resort, to “medium”.

8.2.60.2. Usage:

```
<ctrl:setStyle Attributes/>
```

8.2.60.3. Attributes:

`value="style"`	The current user’s new style. [In; optional; accepts a JSTL expression.]

8.2.61. StartRegistration Tag

8.2.61.1. Description:

The tag starts a user registration process by logging in as ANYONE. The DN for ANYONE is obtained from the application initialization parameter "com.siemens.wegMgr.ldap.anyone", the password is stored in the application’s configuration file **password.properties**.

The tag throws a JspException in case the login fails.

8.2.61.2. Usage:

```
<ctrl:startRegistration/>
```

8.2.61.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the login result (String). See the login tag description for details. [Out; optional; accepts a JSTL expression.]

8.2.62. Token Tag

8.2.62.1. Description:

The tag splits a string of tokens around matches of a regular expression, and returns or prints the token at a given position. The tag attributes include:

- The string of tokens.
- The regular expression for the token delimiter (as specified in the java.util.regex package).
- The index of the token to be returned or printed.
- The name of the scoped variable to accept the result. If left unspecified, the token is printed to the response writer of the JSP page.

8.2.62.2. Usage:

```
<ctrl:token Attributes/>
```

8.2.62.3. Attributes:

`delim="`*`regularExpression`*`"`	The token delimiter (String). [In; optional; default: “:”; accepts a JSTL expression.]
`item="`*`itemIndex`*`"`	The index of the token to be printed or returned (Integer). [Out; optional; default. “0”; accepts a JSTL expression.]

`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`tokens="`*`tokens`*`"`	The string of tokens. [In; required; accepts a JSTL expression.]
`trim="true"|"false"`	Whether to trim leading and trailing spaces before printing the token. [In; optional; default: “false“; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the operation’s result. [Out; optional; accepts a JSTL expression.]

8.2.63. UnassignPrivilegeFromUsers Tag

8.2.63.1. Description:

Unassigns a privilege from a list of users.

The tag attributes include:

- The DN of the privilege to be unassigned.
- The DNs of the users the privilege is to be unassigned from.
- A flag that controls operation signing.
- init: Don’t perform any unassignment. Just generate and return an XML document to sign the unassignments.
- finish: Validate the provided signed document. Perform the unassignments in the directory, and save the signed document.
- The due date for the unassignment.
- The reason for requesting to withdraw the assignments. The reason is displayed to potential approvers.

The tag returns:

- The XML document to be signed if requested.
- The operation’s result:
- OK: The operation was successful.
- READ_ERROR: The privilege could not be read.
- EMPTY_LIST: The user list is empty.
- UNKNOWN_TYPE: The privilege DN denotes neither a role nor a permission nor a group.

- ASSIGNMENT_ERROR: The privilege could not be unassigned from one or more users.
- INVALID_SIGNATURE: The provided signature could not be validated.
- SAVE_SIGNATURE_ERROR: The provided signature could not be saved.
- Detailed information in case of unassignment errors:
- The DNs of the users the privilege has been unassigned from.
- The DNs of the users the privilege could not be unassigned from.

8.2.63.2. Usage:

`<ctrl:unassignPrivilegeFromUsers *Attributes*/>`

8.2.63.3. Attributes:

`dueDate="due*Date*"`	The unassignment due date. [In; optional; accepts a JSTL expression.]
`errorInfo="*varName*"`	The name of the variable to accept error details in case of result ASSIGNMENT_ERROR: (Object[]) errorInfo[0]: String[]: User DNs of successful unassignments errorInfo[1]: String[]: User DNs of failed unassignments [Out; optional; default: “errorInfo”; accepts a JSTL expression.]
`parameters="*parameters*"`	The parameters for a role unassignment. [In; required; accepts a JSTL expression.]
`privilege="*dn*"`	The DN of the privilege. [In; required; accepts a JSTL expression.]
`reason="*reason*"`	The request reason. [In; optional; accepts a JSTL expression.]
`result="*varName*"`	The name of the variable to accept the operation’s result. [Out; optional; default: “result“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for errorInfo and result; defaults to "page". [Out; optional; default: “page“; accepts a JSTL expression.]

`scopeTextToBeSigned="page"|"request"|"session"|"application"`	The scope of the variable to accept the text to be signed. [Out; optional; default: “page“; accepts a JSTL expression.]
`signedData="`*`signedData`*`"`	The signed XML document. [In; optional; accepts a JSTL expression.]
`signState="init"|"finish"`	Whether to start or finish signing the operation. [In; optional; accepts a JSTL expression.]
`users="`*`users`*`"`	The DNs of the users the privilege is to be unassigned from (String[]). [In; optional; accepts a JSTL expression.]
`varTextToBeSigned="`*`varName`*`"`	The name of the variable to accept the XML document to signed (String). [Out; optional; default: “varTextToBeSigned“; accepts a JSTL expression.]

8.2.64. UpdateSelectionList Tag

8.2.64.1. Description:

When modifying or creating an object, the selection list for one of the object’s attributes may depend on the current value of a master attribute, for example locality may depend on country. Whenever the master attribute value changes, the selection list for the dependent attribute should be updated accordingly.

This tag returns the values for an attribute given the master attributes value.

The tag attributes include:

- The DN of the object that is to be modified.
- The name of the dependent attribute.
- Name and value of the master attribute.

8.2.64.2. Usage:

`<ctrl:updateSelectionList` *`Attributes`*`/>`

8.2.64.3. Attributes:

`masterType="`*`attributeName`*`"`	The master attribute name (String). [In; required; accepts a JSTL expression.]

`masterValue="`*`attributeValue`*`"`	The master attribute value (String). [In; required; accepts a JSTL expression.]
`objectId="`*`dn`*`"`	The DN of the object (String). [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`type="`*`attributeName`*`"`	The name of the attribute whose selection list is to be updated (String). [In; required; accepts a JSTL expression.]
`var="`*`varName`*`"`	The name of the variable to accept the operation’s result. [Out; optional; default: “selectionList”; accepts a JSTL expression.]

8.2.65. UpdateVariables Tag

8.2.65.1. Description:

The tag updates scoped variables after an object has been moved, renamed or deleted.

The tag attributes include:

- The object’s new DN (in case of move and rename only).
- The object’s old DN.
- A comma separated list of regular expressions (as specified in the java.util.regex package) for the names of the variables to be eligible for update.
- The scope of the variables.

A variable is updated if its value matches the object’s old DN:

- The variable’s value is replaced by the object’s new DN if specified (move or rename).
- The variable is removed from its scope in case the new DN is left unspecified (delete).

8.2.65.2. Usage:

`<ctrl:updateVariables` *`Attributes`*`/>`

8.2.65.3. Attributes:

`newDN="dn"`	The object's new DN. [In; optional; accepts a JSTL expression.]
`oldDN="dn"`	The object's old DN. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope of the variables to update. [In; optional; default: "session"; accepts a JSTL expression.]
`varNames="varNames"`	A comma separated list of regular expressions for the names of the variables to be updated. [In; required; accepts a JSTL expression.]

8.3. Expression Tags

The tag library implements expression evaluation extensions.

To use a view tag in a JSP page, include the following tag library directive in the page:

```
<%@ taglib uri="http://www.siemens.com/directory/webManager/expr"
prefix="expr" %>
```

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/webMgrView_1_0.tld** in the **webManager.jar** file, and cannot be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center application is "expr".

8.3.1. AttributeContains Tag

8.3.1.1. Description:

The tag defines an instance of method attributeContains.

8.3.1.2. Usage:

```
<expr:attributeContains Attributes/>
```

8.3.1.3. Attributes:

`caseIgnore="true"|"false"`	Whether to compare ignoring cases (true) or not (false). [In; optional; default: "true"; accepts a JSTL expression.]

`name="name"`	The attribute name; a String. [In; required; accepts a JSTL expression.]
`value="value"`	The attribute value to compare against; a String. [In; required; accepts a JSTL expression.]

8.3.2. AttributeExists Tag

8.3.2.1. Description:

The tag defines an instance of method attributeExists.

8.3.2.2. Usage:

```
<expr:attributeExists Attributes/>
```

8.3.2.3. Attributes:

`name="name"`	The attribute name; a String. [In; required; accepts a JSTL expression.]

8.3.3. AttributeMatches Tag

8.3.3.1. Description:

The tag defines an instance of method attributeMatches.

8.3.3.2. Usage:

```
<expr:attributeMatches Attributes/>
```

8.3.3.3. Attributes:

`all="true"|"false"`	Whether all attribute values must match (true) or one match suffices (false). [In; optional; default: “false”; accepts a JSTL expression.]
`caseIgnore="true"|"false"`	Whether to compare ignoring cases (true) or not (false). [In; optional; default: “true”; accepts a JSTL expression.]

`expression="`*`expression`*`"`	The regular expression (see java.util.regex.Pattern) to match against; a String. [In; required; accepts a JSTL expression.]
`name="`*`name`*`"`	The attribute name; a String. [In; required; accepts a JSTL expression.]
`not="true"|"false"`	Whether the pattern must match (false) or not match (true). [In; optional; default: “false”; accepts a JSTL expression.]

8.3.4. Method Tag

8.3.4.1. Description:

The tag defines a name for a method. The method is defined in the tag’s body.

8.3.4.2. Usage:

```
<expr:method Attributes/>
```

8.3.4.3. Attributes:

`name="`*`name`*`"`	The attribute name; a String. [In; required; accepts a JSTL expression.]

8.3.5. Methods Tag

8.3.5.1. Description:

The tag assigns a method map to a scoped variable. The map is defined in the tag’s body

8.3.5.2. Usage:

```
<expr:methods Attributes/>
```

8.3.5.3. Attributes:

`scope="page"|"request"|"session"|"application"`	The scope for the variable. [In; optional; default: “session“; accepts a JSTL expression.]

`var="varName"`	The name of the scoped variable. [In; optional; default: “apply”; accepts a JSTL expression.]

8.4. View Tags

This section describes the usage of the DirXIdentity Web Center view tag library.Its tags are intended to be used in view JSPs but not in action JSPs.

To use a view tag in a JSP page, include the following tag library directive in the page:

<%@ taglib uri="http://www.siemens.com/directory/webManager/view" prefix="view" %>

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/webMgrView_1_0.tld** in the **webManager.jar** file, and cannot be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center application is “view”.

8.4.1. AddTokens Tag

8.4.1.1. Description:

The tag outputs the HTML code for hidden form fields containing the CSRF token and the request ID.

8.4.1.2. Usage:

`<view:addTokens Attributes/>`

8.4.1.3. Attributes:

`format="html"|"javascript"|"asVar"`	The token format. [In; optional; default: “html”; accepts a JSTL expression.]
`page="true"|"false"`	Whether to store the CSRF token for a page refresh (true) or not (false). [In; optional; default: “false”; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for varName and varValue. [In; optional; default: “page“; accepts a JSTL expression.]

`varName="varName"`	The name of the variable to accept the CSRF token name (String). [In; optional; default: “csrfTokenName”; accepts a JSTL expression.]
`varValue="varValue"`	The name of the variable to accept the CSRF token value (String). [In; optional; default: “csrfTokenValue”; accepts a JSTL expression.]

8.4.2. Alert Tag

8.4.2.1. Description:

The tag generates the HTML code for displaying a localized message according to the session-scoped message variables and to internal error flags.

The message-related session-scoped variables are removed, the internal error flags reset.

8.4.2.2. Usage:

```
<view:alert Attributes/>
```

8.4.2.3. Attributes:

`renderer="rendererName"`	The tag renderer. [In; optional; default: “alert”; accepts a JSTL expression.]
`defer="defer"`	Whether to defer message display until the document is loaded. [In; optional; default: “false”; accepts a JSTL expression.]

8.4.3. Assign Tag

8.4.3.1. Description:

The tag generates the HTML code for displaying an assign form and adds it to the response. The form contains an upper (left) table for the items that can be assigned (available items), a lower (right) table for the items that are already assigned, and buttons to move items from top to bottom (left to right) and vice-versa.

8.4.3.2. Usage:

```
<view:assign Attributes>
Body content
```

```
</view:assign>
```

8.4.3.3. Attributes:

`buttonRenderer="`*`rendererName`*`"`	The renderer for the assign buttons. [In; optional; default: “submit”; accepts a JSTL expression.]
`componentRenderer="`*`rendererName`*`"`	The renderer for the assign control. [In; optional; default: “assignh” (horizontal mode) or “assignv” (vertical mode); accepts a JSTL expression.]
`height="`*`height`*`"`	The height of the assign control. [In; optional; default: ”200”; accepts a JSTL expression.]
`labelRenderer="`*`rendererName`*`"`	The label renderer for the assign control. [In; optional; default: “label”; accepts a JSTL expression.]
`mode="horizontal"|"vertical"`	Whether to align the tables horizontally or vertically. [In; optional; default: “horizontal”; accepts a JSTL expression.]
`readonly="true"|"false"`	Whether to show (“false”) or hide (“true”) the available items. [In; optional; default: “false”; accepts a JSTL expression.]
`rendererId="`*`uniqueIdentifier`*`"`	The unique identifier of the assign control. [Required; accepts a JSTL expression.]
`size="`*`size`*`"`	The page size (maximum number of items per page) for the item tables. [In; optional; accepts a JSTL expression.]

8.4.3.4. Body Content:

The tag accepts any JSP content in its body, especially the following child tags:

`available`	The items available for being assigned.
`assigned`	The items already assigned to the object.

8.4.4. Assigned Tag

8.4.4.1. Description:

The tag generates the HTML code for the assigned items table in an assign form and adds it to the response.

8.4.4.2. Usage:

`<view:assigned <i>Attributes</i>>`

8.4.4.3. Attributes:

`data="`*`data`*`"`	The assigned items (DirectoryEntryBean[]). [In; required; accepts a JSTL expression.]
`renderer="`*`rendererName`*`"`	The tag renderer. [In; optional; default: the corresponding renderer from defaultRenderer.properties; accepts a JSTL expression.]

8.4.4.4. Ancestor Tag:

`assign`	The assign form control.

8.4.5. Available Tag

8.4.5.1. Description:

The tag generates the HTML code for the available items (the items that can be assigned) table in an assign form and adds it to the response.

8.4.5.2. Usage:

`<view:available <i>Attributes</i>/>`

8.4.5.3. Attributes:

`data="`*`data`*`"`	The available items (DirectoryEntryBean[]). [In; required; accepts a JSTL expression.]
`limitWarning="true"|"false"`	Whether a time or size limit warning should be displayed. [In; optional; default: “false”; accepts a JSTL expression.]

`noHitWarning="true"|"false"`	Whether a no-hit warning should be displayed. [In; optional; default: “false”; accepts a JSTL expression.]
`renderer="`*`rendererName`*`"`	The tag renderer. [In; optional; default: the corresponding renderer from defaultRenderer.properties; accepts a JSTL expression.]

8.4.5.4. Ancestor Tag:

`assign`	The assign form control.

8.4.6. Expression Tag

8.4.6.1. Description:

The tag recursively evaluates a JSTL expression as long as the result is of type String and includes a JSTL expression. The final result is either returned in a scoped variable, or turned into a String (by applying the toString method) and printed to the response writer of the JSP page.

The tag attributes include:

- The expression to be evaluated.
- The expected Java class of the final result.
- The maximum number of times the expression is recursively evaluated. Expression evaluation stops if the result of the previous evaluation step doesn’t contain any expressions.
- The default value if the expression evaluates to null or to an empty String. If left unspecified, the result will be null or the empty String.
- The result variable’s name and scope. If the name is left unspecified, the result is handed to the response writer.

8.4.6.2. Usage:

`<view:expression` *`Attributes`*`/>`

8.4.6.3. Attributes:

`default="`*`default`*`"`	The default value. [In; optional; accepts a JSTL expression.]

`iterations="iterations"`	The maximum number of times the expression is recursively evaluated. [In; optional; default: “10“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`type="className"`	The expected Java class of the evaluation result. [In; optional; default: “java.lang.String”; accepts a JSTL expression.]
`value="value"`	The expression to be evaluated. [In; required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the operation’s result. [Out; optional; accepts a JSTL expression.]

8.4.7. File Tag

8.4.7.1. Description:

The tag reads a text file from the server’s disk and prints the file content to the response of the current JSP page.

The tag is used to send files like generated reports to the client in order to display the file content in an Iframe or to save it to the client’s disk.

The tag attributes include:

- The server file name.
- The server file encoding like “UTF-8” or “ISO-8859-1” (For details, see the Java package java.nio.charset).
- The buffer size for reading the file.

8.4.7.2. Usage:

```
<view:file Attributes/>
```

8.4.7.3. Attributes:

`bufferSize="bufferSize"`	The buffer size for reading the file content. [In; optional; default: “8192”; accepts a JSTL expression.]

`contextRelative="false"|"true"`	Whether (**true**) or not (**false**) the file name is relative to the Web application context. [In; optional; default: “false”; accepts a JSTL expression.]
`encoding="`*`charSet`*`"`	The file encoding (String). [In; optional; default: the server’s default encoding; accepts a JSTL expression.]
`name="`*`fileName`*`"`	The name of the server file. [In; required; accepts a JSTL expression.]

8.4.8. Focus Tag

8.4.8.1. Description:

The tag sets and/or resets the focus settings. This is used to set the focus properly when displaying a page.

8.4.8.2. Usage:

`<view:focus` *`Attributes`*`/>`

8.4.8.3. Attributes:

`nodeId="`*`nodeId`*`"`	Sets the id of the DOM node to assign the focus to. [In; optional; accepts a JSTL expression.]
`reset="false"|"true"`	Reset the focus settings. [In; optional; default: “false”; accepts a JSTL expression.]
`toolClass="`*`toolClass`*`"`	Sets the item CSS class of the tool bar to assign the focus to. [In; optional; accepts a JSTL expression.]

8.4.9. Form Tag

8.4.9.1. Description:

The tag generates the HTML code for a form configured in a **forms-config.xml** file and adds it to the response. The code renders the form and all its elements.

The tag attribute action accepts the path of a Struts action (for example, “/storeUserData.do”). The action is used in two ways:

- The form to be generated is the one assigned to that action.
- The value for the action attribute of the generated HTML form tag is the action path, prefixed with the application context path (for example, “/webCenter/storeUserData.do”).

8.4.9.2. Usage:

`<view:form Attributes/>`

8.4.9.3. Attributes:

`action="actionPath"`	The action path for the form. [In; required; accepts a JSTL expression.]
`enctype="formEncoding"`	The form encoding to be used. [In; optional; accepts a JSTL expression.]
`method="get"|"post"`	The HTTP method used to submit the form. [In; optional; default: “post”; accepts a JSTL expression.]
`name="formName"`	The form bean name. [In; optional; accepts a JSTL expression.]
`onsubmit="handlerCode"`	The Javascript handler code for form submit events. [In; optional; accepts a JSTL expression.]
`scope="request"|"session"`	The scope for the form. [In; optional; accepts a JSTL expression.]
`style="style"`	Value for the form’s CSS style attribute. [In; optional; accepts a JSTL expression.]
`styleClass="styleClass"`	The form’s CSS style name. [In; optional; accepts a JSTL expression.]
`type="type"`	The form bean type, ie it’s Java class name. [In; optional; accepts a JSTL expression.]

8.4.9.4. Body Content:

The form tag extends org.apache.struts.taglib.html.FormTag. It accepts the same body content as the Struts tag, e.g. tiles:insert tags.

8.4.10. Insert Tag

8.4.10.1. Description:

The tag extends the tiles:insert tag but accepts a JSTL expression for the page name.

8.4.10.2. Usage:

```
<view:insert Attributes/>
```

8.4.10.3. Attributes:

`page="page"`	The path of the page to insert. [In; required; accepts a JSTL expression.]

8.4.11. Language Tag

8.4.11.1. Description:

The tag is used to display the selection boxes for language and font size in the header section of the default Web Center applications.

The tag attributes include:

- The unique XML identifier for the tag.
- The key of the message that contains the selection options. The options are delimited by semicolons. Each option consists of the option value and the language-dependent display name, delimited by a colon. The option with value “default” specifies the selection box header.
- The renderer for the selection box.
- The name of the Struts action to be called on value changes. The default action is defined by the respective renderer property.

8.4.11.2. Usage:

```
<view:language Attributes/>
```

8.4.11.3. Attributes:

`action="actionName"`	The Struts action name (String). [In; optional; default: value for renderer attribute “action”; accepts a JSTL expression.]

`languages="`*`messageKey`*`"`	The key of the message containing the selection options. [In; optional; default: “java.lang.String”; accepts a JSTL expression.]
`renderer="`*`rendererName`*`"`	The id of the renderer for the combo box. [In; required; accepts a JSTL expression.]
`rendererId="`*`uniqueIdentifier`*`"`	The unique XML identifier for the selection element (String). [In; required.]

8.4.12. Log Tag

8.4.12.1. Description:

The tag writes a log message or detailed information about the current HTTP request or session to the log file.

8.4.12.2. Usage:

`<view:log` *`Attributes`*`/>`

8.4.12.3. Attributes:

`level="error"|"info"|"debug"`	The log level. [In; optional; default: “debug“; accepts a JSTL expression.]
`message="`*`message`*`"`	The log message. [In; optional; accepts a JSTL expression.]
`request="true"|"false"`	Whether to log detailed information about the HTTP request (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]
`session="true"|"false"`	Whether to log detailed information about the HTTP session (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]

8.4.13. Match Tag

8.4.13.1. Description:

The tag if a regular expression matches an input string.

The tag attributes include:

- The input string.
- The regular expression (as specified in the java.util.regex package).
- A flag to request a case-insensitive match.

The tag returns:

- A flag indicating whether the input string matched the expression.
- The input substrings that were captured by the expressions capturing groups.

8.4.13.2. Usage:

`<view:match` *`Attributes`*`/>`

8.4.13.3. Attributes:

`caseIgnore="true"|"false"`	Whether to perform a case-insensitive match. [In; optional; default: “false”; accepts a JSTL expression.]
`pattern="`*`pattern`*`"`	The regular expression. [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for varGroups and varMatch. [In; optional; default: “page“; accepts a JSTL expression.]
`value="value"`	The input string. [In; optional; default: the empty string; accepts a JSTL expression.]
`varGroups="`*`varName`*`"`	The name of the variable to accept input subsequences captured by the pattern’s capturing groups (String[]). [In; optional; default: “groups”; accepts a JSTL expression.]
`varMatch="`*`varName`*`"`	The name of the variable to accept the match result (Boolean). [Out; optional; default: “match”; accepts a JSTL expression.]

8.4.14. Menu Tag

8.4.14.1. Description:

The tag generates the Javascript code for the menu bar.

The tag attributes include:

- The name of the menu bar's Tiles definition. The generated menu bar contains only those menus and items the logged in user is granted access to.
- The id of the HTML element the menu bar is to be displayed in. The HTML page must contain an element with that id. The current Javascript implementation assumes that the HTML element is a table row (<tr>).

8.4.14.2. Usage:

```
<view:menu Attributes/>
```

8.4.14.3. Attributes:

`definition="definition"`	The menu bar's tiles definition name. [In; required; accepts a JSTL expression.]
`ifModified="true"|"false"`	Whether to generate the HTML code for the menu only if the new menu bar is different from the previously generated one. [In; optional; default: "false"; accepts a JSTL expression.]
`menubarId="menubarId"`	The ID of the HTML element in which to display the menu bar (String). [In; required; accepts a JSTL expression.]
`vertical="true"|"false"`	Whether to generate code for a vertical or horizontal menu. [In; optional; default: "false"; accepts a JSTL expression.]

8.4.15. NavigationHistory Tag

8.4.15.1. Description:

The tag generates the Javascript code to set the options for the navigation history select element.

8.4.15.2. Usage:

```
<view:navigationHistory Attributes/>
```

8.4.15.3. Attributes:

`rendererId="`*`rendererId`*`"`	The ID of the Web Center renderer to generate the Javascript code. [In; required; accepts a JSTL expression.]

8.4.16. Out Tag

8.4.16.1. Description:

The tag escapes special characters in a string and writes the result to the HTTP response. The tag provides a variety of escape algorithms for inserting strings into various parts of HTML, XML and Javascript code.

The tag attributes include:

- The value to escape and write to the response.
- The name of the escape algorithm:
- **html**: Escape special HTML characters.
- **htmlId**: Convert the input string to a valid HTML id. Replaces invalid characters with underscores and then removes leading underscores.
- **htmlPreserveNewLines**: Escape special HTML characters and replace each new line character with the HTML tag “
`”.
- **js**: Escape special Javascript characters.
- **jsd**: Escape special Javascript characters for Javascript code enclosed in double quotes.
- **jss**: Escape special Javascript characters for Javascript code enclosed in single quotes.
- **jssd**: Escape special Javascript characters for Javascript code enclosed in inner single quotes and outer double quotes.
- **xml**: Escape special XML characters.

8.4.16.2. Usage:

`<view:out` *`Attributes`*`/>`

8.4.16.3. Attributes:

`value="`*`value`*`"`	The value to escape and write. [In; optional; accepts a JSTL expression.]
`escape="html"|"htmlId"|"htmlPreserveNewLines"|` `"js"|"jsd"|"jss"|"jssd"|"xml"`	The escape algorithm. [In; optional; default: “html”; accepts a JSTL expression.]

`recursiveExpressionEvaluation="true"|"false"`	Whether (**true**) or not (**false**) to evaluate expressions in the value recursively. [In; optional; default: “false”; accepts a JSTL expression.]

8.4.17. Scripts Tag

8.4.17.1. Description:

The tag generates the HTML script tags to include a list of Javascripts.

Script lists are defined in **webCenter.properties**. Each script list has an identifier; its value is a comma-separated list of path names of Javascript files.

The tag takes the script identifier from a request-scoped variable, and returns the script list identifier and the generated HTML script tags in scoped variables.

8.4.17.2. Usage:

`<view:scripts` *`Attributes`*`/>`

8.4.17.3. Attributes:

`listParam="`*`listParam`*`"`	The name of the request parameter holding the id of the script list to include. Default: “scriptListId”. [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for varId and varTags. [In; optional; default: “page“; accepts a JSTL expression.]
`varId="`*`varName`*`"`	The script list identifier (String). [Out; optional; default: “scriptListId”; accepts a JSTL expression.]
`varTags="`*`varName`*`"`	The generated script tags (String). [Out; optional; default: “scriptTags”; accepts a JSTL expression.]

8.4.18. SearchPanel Tag

8.4.18.1. Description:

The tag adds the HTML code for a search panel to the response.

The tag attributes include

- The search filter attributes:
  <LDAP attribute name>:<Message key>;...
- The search filter operands:
  <Operand name>:<Message key>;...

  The supported operand names are approx, beginsWith, contains, endsWith, equals, greaterOrEqual, isPresent and lessOrEqual, and their negated counterparts, like not.contains. But note that some operands will work only for specific attributes.
- The search filter object classes:
  <LDAP objectclass name>:<Message key>;...
- The number of filter rows; each row displays the same attributes and operands.

A search can be performed synchronously or asynchronously. In the latter case, the search response does not replace the entire search page but only the search result list. The id of the result table's parent element must be supplied in attribute "targetId"; if the id is left unspecified, the whole page is rewritten.

8.4.18.2. Usage:

```
<view:searchPanel Attributes/>
```

8.4.18.3. Attributes:

`attributes="attributes"`	The search filter attributes. [In; required; accepts a JSTL expression.]
`buttonLabel="messageKey"`	The label for the search button. [In; optional; accepts a JSTL expression.]
`buttonTitle="messageKey"`	The title (tooltip) for the search button. [In; optional; accepts a JSTL expression.]
`conjunctions="and;or"|"or;and"|"and"|"or"`	The filter conjunctions. In case both conjunctions are specified, the first one is the default conjunction. [In; required; accepts a JSTL expression.]
`criteriaCount="criteriaCount"`	The number of filter rows. [In; optional; default: "1"; accepts a JSTL expression.]
`filter="ldapFilter"`	The initial filter input. [In; optional; accepts a JSTL expression.]
`filterLabel="messageKey"`	The label for the search filter. [In; optional; accepts a JSTL expression.]

`hideSearchBase="true"|"false"`	Whether to show or hide the search base input fields. [In; optional; default: “false”; accepts a JSTL expression.]
`hideTitle="true"|"false"`	Whether to show or hide the search panel title. [In; optional; default: “false”; accepts a JSTL expression.]
`includeTypes="`*`includeTypes`*`"`	A semi-colon separated list of additional object types that may be included in the search operations. Each type starts with the mandatory type name, optionally followed by the name of a scoped variable indicating whether the type is enabled, and the message label key. [In; optional; accepts a JSTL expression.]
`objectClasses="`*`ldapObjectClasses`*`"`	The object classes for the object class selection box. [In; optional; accepts a JSTL expression.]
`objectClassLabel="`*`messageKey`*`"`	The label for the object class selection box. [In; optional; accepts a JSTL expression.]
`operands="`*`operands`*`"`	The filter operands. [In; required; accepts a JSTL expression.]
`renderer="`*`rendererName`*`"`	The search panel renderer. [In; optional; default: “searchPanel”; accepts a JSTL expression.]
`rendererId="`*`uniqueIdentifier`*`"`	The unique identifier for the search panel. [In; required; accepts a JSTL expression.]
`searchBase="`*`dn`*`"`	The initial search base. [In; required; accepts a JSTL expression.]
`searchBaseLabel="`*`messageKey`*`"`	The label for the search base field. [In; optional; accepts a JSTL expression.]
`showObjectClassSelector="true"|"false"`	Whether to show or hide the object class selection box. [In; optional; default: “false”; accepts a JSTL expression.]

`targetId="targetId"`	The id of the HTML element whose content is to be replaced by the tag's output. [In; optional; accepts a JSTL expression.]
`title="title"`	The search panel title. [In; optional; accepts a JSTL expression.]

8.4.19. Tab Tag

8.4.19.1. Description:

The tag generates the HTML code for a tab in a tab sheet and adds it to the response. The action to be executed on clicking the tab is either identified by its forward name or its action path. Within a single tab sheet, all tabs must use the same method to identify the action.

8.4.19.2. Usage:

`<view:tab Attributes/>`

8.4.19.3. Attributes:

`Forward="forward"`	The tab's forward name. [In; optional; accepts a JSTL expression.]
`label="label"`	The tag label or its message key. [In; required; accepts a JSTL expression.]
`path="path"`	The tab's action path. [In; optional; accepts a JSTL expression.]

8.4.19.4. Ancestor Tag:

`tabSheet`	The tab sheet for this tab.

8.4.20. Table Tag

8.4.20.1. Description:

The tag generates the HTML code for a scrollable table and adds it to the response. The table must be configured as a property of the current action form.

A table is part of an HTML form. It can, however, update its content asynchronously without a simultaneous update of its enclosing form. Since, in that case, the table tag is not enclosed by a form tag, the form name must be provided explicitely in attribute "form".

8.4.20.2. Usage:

`<view:table Attributes/>`

8.4.20.3. Attributes:

`data="propertyName"`	The form property name of the table. [In; required; accepts a JSTL expression.]
`form="formName"`	The name of the form containing the table. [In; optional; accepts a JSTL expression.]
`height="height"`	The table height. [In; optional; default: from form configuration; accepts a JSTL expression.]
`limitWarning="true"|"false"`	Whether to display a message that a time or size limit has been exceeded when retrieving the table data. [In; optional; default: “false”; accepts a JSTL expression.]
`listPageSizeItems="listPageSizeItems"`	The name of the page size selection list. [In; optional; accepts a JSTL expression.]
`listToolbar="listToolbar"`	The Tiles definition name for the list toolbar. [In; optional; accepts a JSTL expression.]
`multiple="true"|"false"`	Whether the table allows for multiple selections. [In; optional; default: “false”; accepts a JSTL expression.]
`noHitWarning="true"|"false"`	Whether to display a message that no entry was found when retrieving the table data. [In; optional; default: “false”; accepts a JSTL expression.]
`renderer="rendererName"`	The table renderer. [In; optional; default: from form configuration; accepts a JSTL expression.]
`rendererId="uniqueIdentifier"`	The table name. [In; required; accepts a JSTL expression.]

`selectedItemsAttribute="propertyName"`	The form property name of the selected items attribute. [In; optional; default: from form configuration; accepts a JSTL expression.]
`selectedRows="selectedRows"`	The row numbers of selected items. [In; optional; default: value of form property selectedItemsAttribute; accepts a JSTL expression.]
`size="size"`	The table page size. [In; optional; default: renderer page size; accepts a JSTL expression.]
`title="title"`	The table caption. [In; optional; default: from form configuration; accepts a JSTL expression.]
`width="width"`	The table width. [In; optional; default: from form configuration; accepts a JSTL expression.] accepts a JSTL expression.]

8.4.21. TabSheet Tag

8.4.21.1. Description:

The tag generates the HTML code for a tab sheet and adds it to the response.

The tab to be displayed is either identified by its forward name or its action path, depending on whether the included tab tags use forward names or action paths.

8.4.21.2. Usage:

`<view:tabSheet Attributes>`

Body content

</view:tabSheet>

8.4.21.3. Attributes:

`action="uniqueIdentifier"`	The action to be performed when the tab is activated. [In; required; accepts a JSTL expression.]

`height="height"`	The tab sheet height. [In; optional; accepts a JSTL expression.]
`onTabChange="handlerCode"`	The Javascript handler code for tab change events. [In; optional; accepts a JSTL expression.]
`renderer="rendererName"`	The tab sheet renderer. [In; optional; default: "tabSheet"; accepts a JSTL expression.]
`rendererId="uniqueIdentifier"`	The tab sheet name. [In; required]
`selectedForward="selectedForward"`	The forward name of the tab to be displayed. [In; optional; accepts a JSTL expression.]
`selectedPath="selectedPath"`	The forward path of the selected tab to be displayed. [In; optional; accepts a JSTL expression.]
`width="width"`	The tab sheet width. [In; optional; accepts a JSTL expression.]

8.4.21.4. Body Content:

The tag accepts any JSP content in its body, especially the following child tags:

`tab`	The tabs for this tab sheet.

8.4.22. Token Tag

8.4.22.1. Description:

The tag splits a string of tokens around matches of a regular expression, and returns or prints the token at a given position. The tag attributes include:

- The string of tokens.
- The regular expression for the token delimiter (as specified in the java.util.regex package).
- Whether to discard leading and trailing spaces of the tokens.
- The index of the token to be returned or printed.
- The name of the scoped variable to accept the result. If left unspecified, the token is printed to the response writer of the JSP page.

8.4.22.2. Usage:

```
<view:token Attributes/>
```

8.4.22.3. Attributes:

`delim="regularExpression"`	The token delimiter (String). [In; optional; default: ":"; accepts a JSTL expression.]
`item="itemIndex"`	The index of the token to be printed or returned (Integer). [Out; optional; default. "0"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`tokens="tokens"`	The string of tokens. [In; required; accepts a JSTL expression.]
`trim="true"|"false"`	Whether to trim leading and trailing spaces (true) or not (false). [In; optional; default: "false"; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the operation's result. [Out; optional; accepts a JSTL expression.]

8.4.23. Toolbar Tag

8.4.23.1. Description:

The toolbar tag reads the Tiles definition of a toolbar and returns the list of toolbar items. It also generates an application-wide unique toolbar id.

8.4.23.2. Usage:

```
<view:toolbar Attributes/>
```

8.4.23.3. Attributes:

`definition="definition"`	The Tiles definition name of the toolbar. [In; optional; accepts a JSTL expression.]

`prefix="prefix"`	The prefix for the toolbar id. [In; optional; default: “tb“; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for var and varId. [In; optional; default: “page“; accepts a JSTL expression.]
`toolbarId="toolbarId"`	The toolbar id. [In; optional; default: an automatically generated identifier; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the toolbar item array. [Out; optional; default: “toolbar“; accepts a JSTL expression.]
`varId="varName"`	The name of the variable to accept the toolbar id. [Out; optional; default: “toolbarId“; accepts a JSTL expression.]

8.5. Workflow Tags

This section describes the usage of the DirXIdentity Web Center workflow tag library.Its tags are intended to be used in action JSPs to handle workflows and activities.

To use a workflow tag in a JSP page, include the following tag library directive in the page:

<%@ taglib uri="http://www.siemens.com/directory/webManager/workflow" prefix="wf" %>

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/webMgrWf_1_0.tld** in the **webManager.jar** file, and may not be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center application is “wf”.

8.5.1. ApproveActivities Tag

8.5.1.1. Description:

The tag approves (accepts or rejects) one or more approval activities (also called bulk approval).

The tag attributes include:

- **activityDNs**: The DNs of the activities to be approved.
- **updateType**: Whether to accept (“grant”) or reject (“deny”) the approval requests.

- **reason**: The reason for accepting or rejecting the requests.

The tag attribute **result** indicates the outcome of the operation:

- The string “OK” if the approvals succeeded.
- An error message text in case one or more approvals failed.

In any case, the tag also returns

- **numSucceededApprovals**: The number of successfully approved activities.
- **numFailedApprovals**: The number of activities which failed to be approved.

In case of error, the tag additionally returns

- **failedApprovals**: The activity DNs of the approvals which failed.

8.5.1.2. Usage:

`<wf:approveActivities` *`Attributes`*`/>`

8.5.1.3. Attributes:

`activityDNs="`*`dns`*`"`	The activity DNs (String[]). [In; required; accepts a JSTL expression.]
`failedApprovals="`*`varName`*`"`	The name of the variable to accept the DNs of the failed approvals (String[]). [Out; optional; default: “failedApprovals”; accepts a JSTL expression.]
`numFailedApprovals="`*`varName`*`"`	The name of the variable to accept the number of failed approvals (Integer). [Out; optional; default: “numFailedApprovals”; accepts a JSTL expression.]
`numSucceededApprovals="`*`varName`*`"`	The name of the variable to accept the number of successful approvals (Integer). [Out; optional; default: “numSucceededApprovals”; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for result. [In; optional; default: “page“; accepts a JSTL expression.]

`reason="reason"`	The reason for approving or rejecting the approval activities. [In; optional; accepts a JSTL expression.]
`result="varName"`	The name of the variable to accept the operation's result. [Out; optional; default: "result"; accepts a JSTL expression.]
`updateType="deny"|"grant"`	The update type. [In; required; accepts a JSTL expression.]

8.5.2. CreateInstance Tag

8.5.2.1. Description:

The tag creates a workflow instance for a given workflow definition and returns its ID.

8.5.2.2. Usage:

```
<wf:createInstance Attributes/>
```

8.5.2.3. Attributes:

`definition="dn"`	The workflow definition name. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for the result variable. [In; optional; default: "page"; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the result (String). [Out; required; accepts a JSTL expression.]

8.5.3. GetActivity Tag

8.5.3.1. Description:

The tag returns the activity with a given DN.

8.5.3.2. Usage:

```
<wf:getActivity Attributes/>
```

8.5.3.3. Attributes:

`activityDN="dn"`	The activity DN. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for var. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeDueDateEnabled="page"|"request"|"session"|"application"`	The scope for varDueDateEnabled. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeReadOnly="page"|"request"|"session"|"application"`	The scope for varReadOnly. [In; optional; default: “page“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the result (RequestActivityInstance). [Out; required; accepts a JSTL expression.]
`varDuedateEnabled="varName"`	The name of the variable to accept the due date enabled flag of the activity (Boolean). [Out; optional; default: “dueDateEnabledActivity”; accepts a JSTL expression.]
`varReadOnly="varName"`	The name of the variable to accept the read-only flag of the activity (Boolean). [Out; optional; default: “readonlyActivity”; accepts a JSTL expression.]

8.5.4. GetDefinitions Tag

8.5.4.1. Description:

The tag returns the list of workflow definitions for a given subject (like dxrUser or dxrRole) and of a given type (like create).

8.5.4.2. Usage:

```
<wf:getDefinitions Attributes/>
```

8.5.4.3. Attributes:

`exclude="exclude"`	Regular expressions for workflow definition name, description or path. Definitions matching the expressions are excluded from the list. [In; optional; accepts a JSTL expression.]
`include="include"`	Regular expressions for workflow definition name, description or path. Definitions not matching the expressions are excluded from the list. [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]
`subject="subject"`	The workflow definition subject. [In; required; accepts a JSTL expression.]
`type="type"`	The workflow definition type. [In; required; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the workflow definition list (RemoteWorkflowDefinition[]). [Out; required; accepts a JSTL expression.]

8.5.5. GetInstanceId Tag

8.5.5.1. Description:

The tag returns the workflow instance ID from an activity DN.

8.5.5.2. Usage:

```
<wf:getInstanceId Attributes/>
```

8.5.5.3. Attributes:

`dn="dn"`	The activity DN. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application`	The scope for the result variable. [In; optional; default: “page“; accepts a JSTL expression.]

`var="varName"`	The name of the variable to accept the workflow instance ID (String). [Out; required; accepts a JSTL expression.]

8.5.6. GetNextInteractiveActivity Tag

8.5.6.1. Description:

The tag returns the next interactive running activity for a given workflow instance.

The workflow server cannot determine the next interactive activity as long as any (interactive or automatic) activity for the workflow is currently executed. In this case, you can have the server wait for the workflow to become idle by defining a positive idle timeout. If the workflow is still busy when the timeout expires, the idle flag will be set to “TRUE”.

8.5.6.2. Usage:

```
<wf:getNextInteractiveActivity Attributes/>
```

8.5.6.3. Attributes:

`instanceId="id"`	The workflow instance id. [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for var. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeActivityDN="page"|"request"|"session"|"application"`	The scope for varActivityDN. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeDueDateEnabled="page"|"request"|"session"|"application"`	The scope for varDueDateEnabled. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeIdle="page"|"request"|"session"|"application"`	The scope for varIdle. [In; optional; default: “page“; accepts a JSTL expression.]
`scopeReadOnly="page"|"request"|"session"|"application"`	The scope for varReadOnly. [In; optional; default: “page“; accepts a JSTL expression.]

`timeout="timeout"`	The maximum time to wait for the workflow instance to become idle (in milliseconds). [In; optional; default: “0”; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the activity (RequestActivityInstance). [Out; required; accepts a JSTL expression.]
`varActivityDN="varName"`	The name of the variable to accept the activity DN (String). [Out; required; accepts a JSTL expression.]
`varDuedateEnabled="varName"`	The name of the variable to accept the due date enabled flag of the activity (Boolean). [Out; optional; default: “dueDateEnabledActivity”; accepts a JSTL expression.]
`varIdle="varName"`	The name of the variable to accept the idle flag (Boolean). [Out; required; accepts a JSTL expression.]
`varReadOnly="varName"`	The name of the variable to accept the read-only flag of the activity (Boolean). [Out; optional; default: “readonlyActivity”; accepts a JSTL expression.]

8.5.7. PopulateListFormBean Tag

8.5.7.1. Description:

The tag populates a form bean with a workflow definition list returned from a listDefinitions call.

8.5.7.2. Usage:

```
<wf:populateListFormBean Attributes/>
```

8.5.7.3. Attributes:

`definitions="definitions"`	The workflow definitions (RemoteWorkflowDefinition[]). [In; required; accepts a JSTL expression.]

`form="form"`	The form bean to be populated (DynaActionForm). [In; required; accepts a JSTL expression.]

8.5.8. UpdateActivity Tag

8.5.8.1. Description:

The tag updates a workflow activity.

The tag attributes include:

- The DN of the activity to be updated.
- The operation type, one of:
- **cancel**: Cancel the activity.
- **assign**: Update the workflow resource orders and the activity. (For request assignment activities only.)
- **changeParticipant**: Change the activity participant.
- **deny**: Update the workflow subject order according to the provided attribute map, set the denial reason and update the activity. (For approval activities only.)
- **enter**: Update the workflow subject order according to the provided attribute map and update the activity. (For enter attribute activities only.)
- **grant**: Update the workflow subject order according to the provided attribute map, set the grant reason and update the activity. (For approval activities only.)
- **password**: Store the new password in the workflow subject order and update the activity. The new password value must be provided in the tag attribute attributes. (For enter password activities only.)
- **saveCertification**: Save the current state of a certification activity.
- **submitCertification**: Save and finish a certification activity.
- A flag controlling operation signing. The flag is evaluated only for the operation types assign, deny, enter and grant.
- **init**: Do not update the activity. Just generate and return an XML document to sign the updates.
- **finish**: Validate the provided signed document. Update the activity, and save the signed document.

In case the update type is "submitCertification", the tag returns

- The string “OK” if the certification task is finished.
- The string “INCOMPLETE_CERTIFICATION” if the certification task is still unfinished.

8.5.8.2. Usage:

`<wf:updateActivity *Attributes*/>`

8.5.8.3. Attributes:

`activityDN="`*`dn`*`"`	The activity DN. [In; required; accepts a JSTL expression.]
`attributes="`*`attributes`*`"`	The workflow subject order password (String) or attributes (Map, Attribute name -> Attribute value). [In; optional; accepts a JSTL expression.]
`dueDate="due`*`Date`*`"`	The due date for enter attributes activities. [In; optional; accepts a JSTL expression.]
`newParticipant="`*`dn`*`"`	In case of change participant: The DN of the new participant. [In; optional; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for result. [In; optional; default: “page“; accepts a JSTL expression.]
`subjectDN="`*`dn`*`"`	The workflow subject DN. [In; optional; accepts a JSTL expression.]
`reason="`*`reason`*`"`	The reason for approving or rejecting an approval activity. [In; optional; accepts a JSTL expression.]
`resources="`*`resources`*`"`	The workflow order resources for approval activities, reflecting changes to assignment attributes by an approver. [In; optional; accepts a JSTL expression.]
`result="`*`varName`*`"`	The name of the variable to accept the operation’s result; for update type submitCertification only. [Out; optional; default: “result”; accepts a JSTL expression.]
`scopeTextToBeSigned="page"|"request"|"session"|"application"`	The scope for varTextToBeSigned. [In; optional; default: “page“; accepts a JSTL expression.]

`signedData="signedData"`	The signed document. [In; optional; accepts a JSTL expression.]
`signState="init"|"finish"`	Whether to start or finish activity update signing. [In; optional; accepts a JSTL expression.]
`updateType="assign"|"cancel"|"changeParticipant"|"deny"|"enter"|"grant"|"password"|"saveCertification"|"submitCertification"`	The update type. [In; required; accepts a JSTL expression.]
`varTextToBeSigned="varName"`	The name of the variable to accept the document to signed (String). [Out; optional; accepts a JSTL expression.]

8.5.9. UpdateWorkflow Tag

8.5.9.1. Description:

The tag cancels, suspends or resumes a workflow instance.

The workflow server cannot update the workflow as long as any (interactive or automatic) activity for the workflow is currently executed. In this case, you can have the server wait for the workflow to become idle by defining a positive idle timeout.

8.5.9.2. Usage:

`<wf:updateWorkflow Attributes/>`

8.5.9.3. Attributes:

`timeout="timeout"`	The maximum time to wait for the update operation to become executable (in milliseconds) [In; optional; default: “0”; accepts a JSTL expression.]
`updateType="cancel"|"resume"|"suspend"`	The update type. [In; required; accepts a JSTL expression.]
`workflowDN="dn"`	The workflow DN. [In; required; accepts a JSTL expression.]

8.6. DirXweb for JSP Tags

This section describes the DirXweb for JSP tags used by Web Center applications.The tag

implementations can be found in jar file DirXweb12.jar.

To use a DirXweb for JSP tag in a JSP page, include the following tag library directive in the page:

```
<%@ taglib uri="/siemens/dirxjsp" prefix="dir" %>
```

Notes

- The specified URI corresponds to the URI used in the tag library descriptor **META-INF/DirXjsp12.tld** in file **DirXjsp12.jar** and cannot be changed.
- The prefix is arbitrary but must be different from any other prefix used in the same page. The prefix used in the standard Web Center applications is “dir”.

8.6.1. Export Tag

8.6.1.1. Description:

The tag discards all output generated within its body.

8.6.1.2. Usage:

```
<dir:export/>
```

8.6.2. GetCookie Tag

8.6.2.1. Description:

The tag splits an HTTP cookie value into sub-cookies. It returns a DirCookie object that provides access methods to the sub-cookies.

8.6.2.2. Usage:

```
<dir:getCookie Attributes/>
```

8.6.2.3. Attributes:

`name="name"`	The cookie name (String). [In; required; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for result. [In; optional; default: “page“; accepts a JSTL expression.]
`var="varName"`	The name of the variable to accept the sub-cookies (DirCookie). [Out; optional; default: “dircookie”; accepts a JSTL expression.]

8.6.3. SetCookie Tag

8.6.3.1. Description:

The tag sets an HTTP cookie created from a set of sub-cookies. Two adjacent sub-cookies will be delimited by a percent sign (%), name and value of a sub-cookie by a dash (-). Sub-cookie values will be Base64-encoded.

8.6.3.2. Usage:

```
<dir:setCookie Attributes>
Body content
</dir:setCookie>
```

8.6.3.3. Attributes:

`name="name"`	The cookie name (String). [In; required; accepts a JSTL expression.]
`path="path"`	The cookie path (String). [In; optional; accepts a JSTL expression.]
`domain="domain"`	The cookie domain (String). [In; optional; accepts a JSTL expression.]
`comment="comment"`	The cookie comment (String). [In; optional; accepts a JSTL expression.]
`httpOnly="true"|"false "`	Whether to set the “httpOnly“ flag for the cookie (true) or not (false). This prevents browser access to the cookie via Javascript code. [In; optional; default: “false“; accepts a JSTL expression.]
`secure="true"|"false "`	Whether to send the cookie over secure protocols only (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]
`maxAge="maxAge"`	The maximum age for the cookie (Integer, in seconds). [In; optional; default: “-1“; accepts a JSTL expression.]
`version="version"`	The cookie version (Integer, 0 or 1). [In; optional; default: “1“; accepts a JSTL expression.]

8.6.3.4. Body Content:

The tag accepts any number of SubCookie tags in its body:

`subCookie`	A sub cookie.

8.6.4. SetParams Tag

8.6.4.1. Description:

The tag is used when uploading files to the web server. It provides easy access to multipart/form-data encoded request parameters. See the use case document on Web Center File Upload for details.

The tag attributes include:

- The comma-separated list of accepted content types provided by the uploading client. If empty, any content type is accepted.
- The comma-separated list of accepted content types returned by the custom content type detector. If empty, any content type is accepted.
- The comma-separated list of accepted file extensions provided by the uploading client. If empty, any file extension is accepted.
- The comma-separated list of accepted mime types returned by the TIKA file type detector. If empty, any mime type is accepted.
- The encoding used to encode string parameters.
- Whether to return the file content or the file items.
- Whether to ignore empty files.
- The maximum total size of data to be uploaded (in bytes).

The tag returns

- A map containing the file parameters.
- A map containing the string parameters.
- An array with information about the operation's result.
- The first item in the array is the result code (Integer:.

  0: The operation was successful.
  1: Size limit exceeded.
  2: Invalid content type.
  3: File upload failed.
  4: Unspecified error.
- The second item is the exception throw in case of errors.

8.6.4.2. Usage:

`<dir:setParams *Attributes*/>`

8.6.4.3. Attributes:

`acceptedContentTypes="acceptedContentTypes"`	The accepted content types (String). [In; optional; accepts a JSTL expression.]
`acceptedCustomContentTypes="acceptedCustomContentTypes"`	The accepted custom content types (String). [In; optional; accepts a JSTL expression.]
`acceptedExtensions="acceptedExtensions"`	The accepted file extensions (String). [In; optional; accepts a JSTL expression.]
`acceptedMimeTypes="acceptedMimeTypes"`	The accepted mime types (String). [In; optional; accepts a JSTL expression.]
`charset="`*`charset`*`"`	The string parameter encoding. [In; optional; default: the platform's default Java encoding; accepts a JSTL expression.]
`content="true"|"false"`	Whether to return the file content (true) or not (false). [In; optional; default: "true"; accepts a JSTL expression.]
`maxFileSize="`*`maxFileSize`*`"`	The maximum upload data size (Integer, in bytes). [In; optional; default: "100000"; accepts a JSTL expression.]
`scope="page"|"request"|"session"|"application"`	The scope for varParam, varFileParam and varResult. [In; optional; default: "page"; accepts a JSTL expression.]
`varParam="`*`varName`*`"`	The name of the variable to accept the string parameters (Map<String,Object>). [Out; optional; default: "dirparam"; accepts a JSTL expression.]
`varFileParam="`*`varName`*`"`	The name of the variable to accept the string parameters (Map<String,Object>). [Out; optional; default: "dirfileparam"; accepts a JSTL expression.]

varResult="*varName*"	The name of the variable to accept the operations result (Object[]). [Out; optional; default: “dirparamresult”; accepts a JSTL expression.]

8.6.5. SubCookie Tag

8.6.5.1. Description:

The tag adds a sub-cookie to the enclosing setCookie tag.

8.6.5.2. Usage:

<dir:subCookie *Attributes*/>

8.6.5.3. Attributes:

name="*name*"	The cookie name (String). [In; required; accepts a JSTL expression.]
path="*path*"	The cookie path (String). [In; optional; accepts a JSTL expression.]
domain="*domain*"	The cookie domain (String). [In; optional; accepts a JSTL expression.]
comment="*comment*"	The cookie comment (String). [In; optional; accepts a JSTL expression.]
httpOnly="true"|"false "	Whether to set the “httpOnly“ flag for the cookie (true) or not (false). This prevents browser access to the cookie via Javascript code. [In; optional; default: “false“; accepts a JSTL expression.]
secure="true"|"false "	Whether to send the cookie over secure protocols only (true) or not (false). [In; optional; default: “false“; accepts a JSTL expression.]
maxAge="*maxAge*"	The maximum age for the cookie (Integer, in seconds). [In; optional; default: “-1“; accepts a JSTL expression.]

`version="`*`version`*`"`	The cookie version (Integer, 0 or 1). [In; optional; default: “1“; accepts a JSTL expression.]

9. Security

This chapter provides information about security.

9.1. Securing Session IDs

9.1.1. Enabling Session Cookies

Session cookies are the preferred way to track session IDs.

For an application running on Tomcat, session cookies are enabled by assigning the value true to attribute cookies in the application's context descriptor file:

```
<Context … cookies="true" …/>
```

Session cookies are by default enabled for the standard Web Center applications.

9.1.2. Disabling URL Rewriting

Tracking session IDs via URL rewriting is considered a security risk and should be turned off.

For Web Center applications, you can turn off URL rewriting by filtering all relevant requests through the RequestFilter, a servlet filter delivered with Web Center.

The filter is defined in the deployment descriptor **web.xml** and mapped to all request URIs ending with *.do or *.jsp and to the URI /saveFile.

The filter is usually configured to disable URL rewriting, since the default value for its initialization parameter URLRewritingEnabled is false.

Since this doesn't cover all possible cases of URL rewriting, also set the session tracking mode to COOKIE in section session-config:

```
<session-config>
    ...
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

9.1.2.1. References

OWASP: Session Management

https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

9.1.3. Setting the HttpOnly Flag

The HttpOnly flag prevents client-side scripts from accessing the session cookie, thereby

mitigating the risk of cross-site scripting attacks.

For an application running on Tomcat, the flag is enabled by assigning the value true to attribute useHttpOnly in the application's context descriptor file:

```
<Context … useHttpOnly="true" …/>
```

The flag is by default enabled for the standard Web Center applications.

9.1.3.1. References

OWASP: HttpOnly

https://www.owasp.org/index.php/HttpOnly

OWASP: Cross-Site Scripting (XSS)

https://www.owasp.org/index.php/XSS

9.2. Preventing Session Fixation

To prevent session fixation attacks, Web Center changes the ID of a user's session after a successful login via single sign-on, via entering user name and password, or via answering challenge/response questions.

A session ID change is triggered by assigning the value true to the request-scoped variable changeSessionID in a logic JSP:

```
<c:set var="changeSessionId" scope="request" value="true"/>
```

The session ID is changed after execution of the logic JSP.

You can enable or disable the feature via a parameter in **webCenter.properties**:

loginChangeSessionId = true

9.2.1.

9.2.1.1. References

OWASP: Session Fixation

https://www.owasp.org/index.php/Session_fixation

9.3. Preventing CSRF Attacks

To prevent cross-site request forgery (CSRF) attacks, Web Center should be configured to require a token to be sent as parameter of each incoming request.The tokens are generated either per request or per session and then stored in the session.If the token of an incoming request matches, the request is accepted.If the token doesn't match or the incoming request does not provide a token at all, the request is (usually) rejected.The

tokens are always transferred as HTTP post parameters so that they are not visible in URLs.You can configure the token's HTTP parameter name, its length and its scope (either request or session).We strongly recommend generating a new token with each request.

This plain approach, however, causes some problems, especially when each request generates a new token.

First, a Struts action often forwards request processing to another one via HTTP redirect.This causes the browser to submit a GET request for the new URL.Since a GET request can only include URL parameters and a CSRF token mustn't be disclosed in a URL, this means that the GET request cannot present any CSRF token at all.Therefore, Web Center stores the redirect URL before sending the response to the original request.The subsequent GET request is then accepted only if its URL matches the stored one.In fact, Web Center stores a list of expected redirect URLs per session to support simultaneous requests which may all get redirected.You can configure the maximum number of entries in the list.When the list size is exceeded the least recently used redirect URL is removed.

Some requests load an HTML page, and then use asynchronous requests to load additional content into certain areas of the page.An example is the Web Center home page which asynchronously loads the content of the plug-ins.Since the requests are sent simultaneously, they all include the same CSRF token, namely the one set by the request loading the HTML page.The first asynchronous request is accepted and changes the token.The other ones will then fail due to a token mismatch.To handle this case as expected Web Center must keep the original token valid for some time.Therefore, Web Center invalidates all tokens that were generated before the token presented by a request.The presented token itself remains valid, and also all tokens generated afterwards, including the new token created when processing the request.

Another problem is caused by page refreshes in the browser.The browser then sends the original request, including the original token.When the tokens vary with each request, the original token is invalid, and the request would usually be rejected.To handle page refreshes as expected by the end user, however, Web Center stores the URLs and the request parameters of the last two requests that loaded an entire HTML page (Requests loading just a part of a page are ignored by browser refreshes.) If an incoming request matches one of the stored requests, it is accepted.

Finally, you can also define a list of application entry points to be exempted from the token match. This allows for bookmarking Web Center pages, and for links from outside into a Web Center application. The list should include only paths that do not compromise security.

The filter is defined in the deployment descriptor **web.xml** and mapped to all request URIs ending with *.do or *.jsp and to the URI /saveFile.

The filter is configured in file **WEB-INF/config/webCenter.properties**:

```
csrf.enabled                 = true
csrf.allowPostForEntryPoints = false
csrf.tokenName               = WT
```

```
csrf.tokenLength              = 20
csrf.tokenScope               = request
csrf.maxRedirectURLs          = 5
csrf.entryPoints              = /login.do,/index.jsp,/error.do,
                                /logout.do
```

To filter is enabled by default.

Notes

- Before you enable the feature it in a productive system test it carefully for possible side-effects.
- Newer versions of Tomcat are shipped with a CSRF prevention filter that uses only one-time tokens.The filter doesn't work with Web Center applications.

9.3.1.

9.3.1.1. References

OWASP: Cross-Site Request Forgery (CSRF)

https://www.owasp.org/index.php/CSRF

9.4. Preventing Clickjacking Attacks

Web Center provides a servlet filter to prevent clickjacking attacks.

Clickjacking attacks may occur if an application runs in a frame or an iframe.The recommended way to prevent such attacks is to restrict framing to the same domain the application was loaded from, or to a trusted domain.The HTTP headers X-Frame-Options and Content-Security-Policy can be used to achieve this.Unfortunately, browser support for these headers is inconsistent.

The filter is defined in the deployment descriptor **web.xml** with name **ClickjackingFilter** and by default mapped to all request URIs.

9.4.1. Allow Framing from Application Domain Only

The filter adds the HTTP header **X-Frame-Options** with value **SAMEORIGIN** to the HTTP response.This instructs browsers to not allow framing Web Center pages from other domains.

```
<filter>
    <filter-name>ClickJackingFilter</filter-name>
    ...
    <init-param>
```

```
        <param-name>X-Frame-Options</param-name>
        <param-value>*:X-Frame-Options:SAMEORIGIN</param-value>
    </init-param>
</filter>
```

The **SAMEORIGIN** option is supported by Internet Explorer, Firefox and Chrome.

9.4.2. Allowing Framing from a Trusted Domain

In this case, the filter adds two HTTP headers to the response.

The **X-Frame-Options** header allows framing from a single URI. It doesn't support wildcards. The header is evaluated by Internet Explorer and Firefox but ignored by Chrome.

The **X-Content-Security-Policy: frame-ancestors** header allows framing from one or more URIs. It supports wildcards. The header is evaluated by Firefox and Chrome but ignored by Internet Explorer.

```
<filter>
    <filter-name>ClickJackingFilter</filter-name>
    ...
    <init-param>
        <param-name>X-Frame-Options</param-name>
        <param-value>*:X-Frame-Options:
     ALLOW-FROM http://my-host.my-company.com:8080
</param-value>
    </init-param>
    <init-param>
        <param-name>X-Content-Security-Policy</param-name>
        <param-value>*:X-Content-Security-Policy:
     frame-ancestors *.my-company.com:*
</param-value>
    </init-param>
</filter>
```

9.4.2.1. References

OWASP: Clickjacking

https://www.owasp.org/index.php/Clickjacking

9.5. Disabling Potentially Risky HTTP Methods

Some HTTP methods can potentially pose a security risk for web applications.Web Center provides a servlet filter that blocks requests with risky methods.

The filter checks the HTTP method of each incoming request.GET and POST requests are always allowed.HEAD, OPTIONS and TRACE requests are optionally allowed.All other ones like PUT and DELETE requests are always rejected.The filter also processes OPTIONS and TRACE requests directly instead of forwarding them further down the processing chain.

The filter is defined in the deployment descriptor **web.xml** with name **MethodFilter** and by default mapped to all request URIs.

The filter is by default configured to reject all requests other than GET and POST requests.To also allow HEAD and OPTIONS requests, for example, add them to the list of allowed methods:

```
<filter>
    <filter-name>MethodFilter</filter-name>
    ...
    <init-param>
        <param-name>AllowedMethods</param-name>
        <param-value>HEAD,OPTIONS</param-value>
    </init-param>
</filter>
```

9.5.1.

9.5.1.1. References

OWASP: Test HTTP Methods (OTG-CONFIG-006)

https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/02-Configuration_and_Deployment_Management_Testing/06-Test_HTTP_Methods

9.6. Configuring Cross-Origin Requests

Traditionally browsers follow a same-origin policy: a web page can send HTTP requests via Javascript only to the server the page was loaded from.HTTP protocol (http/https), server name and port must coincide.

Since this policy was considered too restrictive for some modern types of web applications, the policy was weakened in newer browsers.Now, a browser may send a request to a different server than the origin.The original server is indicated to the request target in a new request header with name “Origin”.The target server can decide whether to accept or

reject the request.Some browsers like Chrome also send “Origin” headers if origin and target server are identical.

By default, Web Center accepts requests with an “Origin” header only if origin and target server coincide.Otherwise, the request is rejected.

You can tell Web Center to accept requests from other origins as well.List the origins to accept in parameter AcceptedCrossOrigins of the RequestFilter in file **web.xml**.Separate the origins by commas.Each origin is evaluated as a case-insensitive regular expression (see Java API documentation of package “java.util.regex”).

The following configuration accepts cross-origin requests from server http://alpha.beta.com:8080. Note that the dots are escaped by backslashes in order to distinguish them from regular expression wildcards.

```
<filter>
  <filter-name>RequestFilter</filter-name>
  ...
  <init-param>
    <param-name>AcceptedCrossOrigins</param-name>
    <param-value>http://alpha\.beta\.com:8080</param-value>
  </init-param>
</filter>
```

9.6.1.

9.6.1.1. References

W3C: Cross-Origin Resource Sharing

http://www.w3.org/TR/cors/

9.7. Input Validation

The flexibility of many components processing Web Center requests is based on JSP expressions.This, however, poses a security risk if end users are allowed to enter JSP expressions into input fields.Therefore, the input of each request should be checked for JSP expressions, and the request should be rejected if an expression is found.JSP expressions start with “${” and end with “}”; examples for input values with JSP expressions are “John ${alpha} Doe” and “${1+2}”.

Web Center’s input validation process performs this check for each incoming request.It parses each request parameter value for JSP expressions.When it detects a JSP expression, it rejects the request.If the request is part of a self-registration, it cancels the self-registration and redirects the user is to the login page.Otherwise, it redirects the user to a previously called page of its session.

You can disable input validation for specific request parameters that are allowed to contain JSP expressions; for example, passwords.The components that subsequently process the request must then make sure that the JSP expressions are not evaluated.

You can configure the request parameters excluded from input validation in the file **webCenter.properties**.You can also disable input validation completely, but this is not recommended.The default configuration is:

```
inputValidation.validateRequestParameters = true
inputValidation.excludedParameters =
         password,oldPassword,retypedPassword,newPassword
```

Adding a parameter to the list of excluded parameters just disables input validation for this parameter.It does not imply that the components involved in processing the request do not evaluate JSP expressions within the parameter value.This must be explicitly ensured and tested in each case.

On detection of a JSP expression, Web Center forwards request processing to the Struts action assigned to the global action forward with name “invalidInput“.By default, this is the action with path “/invalidInput“.The action then either cancels a self-registration or goes back to the last action in the user’s session that is marked as a reset point.

9.8. Error Pages

In order not to disclose any security compromising information to end users and to disrupt user experience as little as possible, Web Center defines some HTML and JSP pages to be displayed in situations where request processing is denied for security reasons or fails due to unrecoverable errors.The pages are configured in the deployment descriptor **web.xml**, for example

```
<error-page>
    <error-code>400</error-code>
    <location>/redirectToError.do</location>
</error-page>
<error-page>
    <error-code>403</error-code>
    <location>/WEB-INF/jsp/view/error/forbidden.html</location>
</error-page>
```

Adapt the definitions and the files in folder **/WEB-INF/jsp/view/error** to your requirements.

9.9. Login Configuration

9.9.1. Login Cookie

Web Center uses a cookie with name login to make some user-specific settings available from one user session to subsequent sessions.The login cookie includes

- The user's Web Center language.
- The user's Web Center font size.
- The value of the user name field of the login form.In case a customized login form contains more than one user identification field, the cookie stores the values of all of them.In case of single sign-on applications, of course, no value is stored at all.

  Note that the cookie is not available to script code in the browser since Web Center sets the cookie's httpOnly flag.

  The cookie's maximum lifetime is by default 30 days.

  If you think that the cookie compromises the security in your environment in any way, you can configure Web Center

- To send the cookie over secure connections only.
- Not to include the login form input in the cookie.
- To set a different maximum age for the cookie.
- To disable the cookie at all.

  The respective configuration parameters (with their default values) are:

loginCookie.enabled = true

loginCookie.includeAttributes = true

loginCookie.maxAge = 2592000

loginCookie.secure = false

For details, see the chapter "Web Center Configuration".

9.9.2. Login Form

9.9.2.1. Disabling Form Authentication

When running a Web Center application with single sign-on, a user is authenticated by some single sign-on component. The component forwards some kind of user credentials to Web Center. Web Center then searches the DirX Identity database for a user matching the presented credentials. This may fail for several reasons: the credentials may match more than one user or a disabled user or no user at all. In case of failure, you can have Web Center either reject the request or display the login form to let the user manually login with name and password. For security reasons, the request should be rejected.

You can enable or disable form login in **webCenter.properties**:

ssoFallbackToLoginPage = false

The default value is “false”.

9.9.2.2. Disabling Autocompletion

Browsers may store values entered into form input fields and redisplay them next time the form is displayed or suggest them as input in a proposal list. This might compromise security since a user may get access to login name and password of a previous user.

Therefore, Web Center deactivates the browser’s autocompletion feature for any password field (in the login form as well as in the forms to change or reset a password). This is achieved by setting the attribute autocomplete to "off" in all password field renderer snippets, for example

```
<input type="password" … autocomplete="off" …
```

When customizing a form with a password field, make sure that autocompletion is deactivated for the password field.

Autocompletion can also be deactivated for the user name field in the login form. The field renderer secureText sets the autocompletion flag to the value of the configuration parameter loginForm.autoComplete. Use this renderer if you define a custom login form with different user name fields.

You can then switch on and off autocompletion for user name fields in **webCenter.properties**:

```
loginForm.autoComplete = off
```

The default value is “off”.

9.9.2.3. Disabling Logins with Incomplete User Names

Another configuration parameter in **webCenter.properties** lets you define the minimum number of characters a user must enter into the user name field of the login form (not counting white spaces). For example, if the minimum number is 3, a login attempt with user name “smi” will succeed if there is only one user whose name starts with “smi” (and of course if the password is correct).

This feature surely compromises security since attackers don’t need a complete user name to break into the application. It must therefore be deactivated in productive environments by assigning the value 0 (which is the default):

```
loginForm.minChars = 0
```

9.9.3. Preventing Non-SSO Requests

When operating Web Center with a single sign-on provider, you can block attempts to access Web Center without single sign-on information.

In file **web.xml**, set parameter ssoRequired of the SSOHeaderFilter to true:

```
<filter>
<filter-name>SSOHeaderFilter</filter-name>
    ...
    <init-param>
        <param-name>ssoRequired</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
```

On receipt of an unauthorized request, Web Center rejects the request. You can instead define an action to redirect the request to.

In file **web.xml**, set parameter ssoLoginAction of the SSOHeaderFilter to the Struts action to redirect to. Note that Web Center does not forward any request parameters to the redirect action.

```
<filter>
<filter-name>SSOHeaderFilter</filter-name>
    ...
    <init-param>
        <param-name>ssoLoginAction</param-name>
        <param-value>/login.do</param-value>
    </init-param>
</filter>
```

9.10. Restricting Access to the Password File

A Web Center application needs one or more passwords for example to access the LDAP server or a Java keystore.The passwords are stored in encrypted format in file **WEB-INF/password.properties**.To change a password, just overwrite its encrypted value with the new clear text value.Then restart Tomcat.During restart, the new password will be encrypted.

Set the password file permissions so that

- The Tomcat server can read and write the file.
- Authorized users can read and write the file in order to change a password.
- No one else can read or write the file.

9.11. Self-Registrations and User ANYONE

To perform user self-registrations, Web Center binds to the directory server for provisioning as user ANYONE (cn=ANYONE,cn=<domain>).User ANYONE is created during initial domain configuration with a default password.Whenever you configure a Web Center application for that domain, the DirX Identity Configurator adds the default password to the application's password file **password.properties**.

The first thing you should do is to change the default password in the directory to a secure value.

Then, if your application uses self-registration:

- Change the password with id "ANYONE" in file **password.properties** accordingly.

If your application does not use self-registration:

- Make sure that file **password.properties** does not contain the correct password.Change the password to any dummy value you like or remove it at all.Now, any self-registration attempt will fail.
- Make sure the self-registration section is not shown on the login page by setting property "loginForm.showRegister" in file **webCenter.properties** to **false** (this is the default value):

loginForm.showRegister = false

- Make sure the self-registration actions "/startRegistration" and "/finishRegistration" in file **WEB-INF/config/workflows/struts-config.xml** are deactivated (which they are by default).Otherwise a user could start a self-registration directly from his browser's address bar.

9.12. Securing Connections

Interprocess communication over an untrusted network should generally be secured via SSL.For a Web Center application, this includes:

- HTTPS between browser and Tomcat.
- LDAP over SSL between Tomcat and the LDAP server.
- SOAP over HTTPS between Tomcat and request workflow service of the Java-based Server.
- JMS over SSL between Tomcat and the message server.

9.12.1. Requiring HTTPS

Web Center can be configured to require HTTPS. Unsecure requests are either redirected to a predefined URL or rejected.

The feature is implemented by the RequestFilter, a servlet filter delivered with Web Center.

It is defined in the deployment descriptor **web.xml** and mapped to all request URIs ending with *.do or *.jsp and to the URI /saveFile.

The filter is by default configured to process any requests whether secure or not. To have insecure requests simply rejected, set its parameter SSLRequired to true:

```
<filter>
    <filter-name>RequestFilter</filter-name>
    ...
    <init-param>
        <param-name>SSLRequired</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
```

To have insecure requests redirected to another page, additionally assign the redirect URL to parameter SSLRedirectURL:

```
<filter>
    <filter-name>RequestFilter</filter-name>
    ...
    <init-param>
        <param-name>SSLRequired</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>SSLRedirectURL</param-name>
        <param-value>
  https://localhost:8443/webCenter-My-Company/login.do
</param-value>
    </init-param>
</filter>
```

In case of a redirection the URL and any parameters of the original request get lost and are not available when processing the redirected request.

9.13. Disabling Development Tools

Web Center provides some tools to help in developing customized applications.In order not to disclose any security compromising information to end users you should deactivate these tools in productive systems.

9.13.1. Development Mode

Turn off Web Center's development mode feature.

In configuration file **WEB-INF/config/webCenter.properties**, set

`developmentMode = false`.

9.13.2. Online Debug

Turn off Web Center's online debug output feature.

In configuration file **WEB-INF/web.xml**, set

```
<context-param>
    <param-name>com.siemens.webMgr.debug.enabled</param-name>
    <param-value>false</param-value>
</context-param>
<context-param>
    <param-name>com.siemens.webMgr.debug.trace.enabled</param-name>
    <param-value>false</param-value>
</context-param>
```

9.14. Output Escaping

When customizing an application, make sure that all data written to an HTTP response is properly escaped in order not to break the generated HTML page or Javascript code.Incorrect escaping also makes a page prone to script code injection attacks by hackers in particular when form input values or request parameters are inserted as part of the response page.

9.14.1. JSPs

Use the JSTL tag <c:out> and the Web Center tag <view:out> to insert data into an HTML page.

9.14.2. Renderer Snippets

Use correct placeholder extensions in HTML and Javascript code snippets.For details, see the section on "Renderer Code Snippets" in chapter "User Interface Configuration".

9.15. Document Root Folder

The document root folder of a web application contains the files that are required to run the application in the web application server.For the standard Web Center applications, the folder contains the file **index.jsp** and the sub folders **resources** and **WEB-INF**.Users can

download any file below the document root provided they know its name, for example via URLs like

`http://host:port/webCenter-domain/index.jsp`

`http://host:port/webCenter-domain/resources/images/new.gif`

`http://host:port/webCenter-domain/resources/build/config/config.js`

The only exception is the **WEB-INF** folder.The application server guarantees that files in the **WEB-INF** folder are not directly accessible to clients.The server also ensures that clients cannot directly access files outside the document root.

Therefore, keep your document root folder clean.Put only those files into the folder (apart from sub folder **WEB-INF**) that must be directly accessible to clients like images, Javascript and VBScript source files, style sheets and static HTML files.Any other file might pose a security risk.

The document root for the standard Web Center applications is *install_path***/web/webCenter-***domain***/webCenter**.The folder contains only the necessary files.

9.16. Securing Tomcat

For recommendations on securing a productive Tomcat server, see the Tomcat documentation and the reference listed here.You may, for example, disable Tomcat's AJP port or enable Tomcat's Security Lifecycle Listener.But note that Web Center does not run with Tomcat Security Manager enabled.

9.16.1.

9.16.1.1. References

OWASP: Securing Tomcat

https://www.owasp.org/index.php/Securing_tomcat

10. Setting up Single Sign-On to the Request Workflow Server

Web Center usually forwards the current user's login credentials to the request workflow server (a component of the Java-based Server) in order to authenticate the user to the server. If you operate Web Center in single sign-on mode, however, the user's password is unknown to Web Center, and hence cannot be used to authenticate the user against the request workflow server. In this case, Web Center authenticates itself to the server with its private key, and forwards only the name of the current user. The request workflow server verifies that it has a valid certificate of the presented private key, and if so, accepts the user name without further validation.

Thus, in case of single sign-on, you need a private key for Web Center, which may be

- A key generated solely for this purpose; see section "Creating a Keystore and a Truststore" below.
- The private key used by Tomcat for HTTPS.

The keystore location and the alias of the private key must be configured in Web Center's deployment descriptor **/WEB-INF/web.xml** in context parameters

- **com.siemens.webMgr.requestworkflow.keystoreName**
- **com.siemens.webMgr.requestworkflow.keyAlias**.

The keystore password and, if different, the key password must be added to Web Center's password file **/WEB-INF/password.properties** with keys

- **webcenterKeystore**
- **webcenterKey**

A certificate of the private key must be imported into the Java-based Server's truststore

- *install_path***/ids-j-***domain***-S***n***/private/webcenter-truststore.**

The truststore password must match the one with key **truststore** in file

- *install_path***/ssl/password.properties.**

The **webcenter-truststore** shares its password with the **server-truststore** of the Java-based Server. When creating a **webcenter-truststore** you must secure it with the **server-truststore** password if you've created the **server-truststore** beforehand. Otherwise, you can choose any password for the **webcenter-truststore** and then assign it manually to key **truststore** in the password file.

If Web Center and the Java-based Server run on different hosts, the keystore must reside on the machine hosting Web Center, the truststore on the one hosting the Java-based Server.

In a domain with multiple Web Center instances you can use the same key for each

instance, or one key per instance, or anything in between. When using different keys, choose a unique alias for each key, and add a certificate for each key to the Java-based Server's **webcenter-truststore**.

In a domain with multiple Java-based Servers, the request workflow server is part of the Java-based Server with request workflow type "default". Since the request workflow engine can be shifted dynamically from one Java-based Server to another, make sure to add the Web Center certificate to the truststores of all affected Java-based Servers.

Note: This chapter applies to single sign-on and to external authentication against an ADS or an LDAP directory. External authentication was first introduced for Web Center for Password Management but works with any Web Center application.

10.1. Creating a Keystore and a Truststore

The batch file

- *install_path***/ids-j-**_domain_**-S**_n_**/utils/ssl/genWebCenter.bat** (or **.sh**)

provides a convenient way to:

- Create a keystore with a private key for Web Center

*install_path***/ids-j-**_domain_**-S**_n_**/private/webcenter-keystore-**_alias_

- Export a self-signed certificate of that key to file

*install_path***/ids-j-**_domain_**-S**_n_**/private/webcenter-**_alias_**.crt**

- Add the certificate to the truststore

*install_path***/ids-j-**_domain_**-S**_n_**/private/webcenter-truststore**

If the truststore doesn't yet exist, it is created. Keystore and certificate file are overwritten if they already exist.

Before running the file, open it and set

- **dname –** A distinguished name intended to identify the Web Center instance(s) the key is generated for.
- **alias -** A unique name for the key.Since the alias gets part of the keystore file name don't use any special characters in the alias.
- **keystorePassword –** The keystore password; any password you like.
- **truststorePassword –** The truststore password.Note that the password is shared with the server-truststore (see previous section).

Afterwards, move the created keystore to the machine hosting Web Center, and adapt the Web Center configuration as described in the previous section.

If you've assigned an initial truststore password change the Java-based Server's

password.properties file accordingly.

When using the batch file again to generate a key for another Web Center instance, choose a different **dname** and **alias**.

10.2. Samples

10.2.1. Web Center and Java-based Server on the Same Host

In the simplest case, you have a single Web Center instance which runs on the same machine as the Java-based Server.

Let's assume

- The folder for the Java-based Server is ***install_path*/ids-j-My-Company-S1.**
- There's isn't yet a server-truststore or a webcenter-truststore for the Java-based Server, so that we can choose the truststore password.

To set up single sign-on, proceed as follows.

10.2.1.1. Generating a Private Key and a Certificate

- Change to folder **install_path/ids-j-My-Company-S1/utils/ssl**.
- Open file **genWebCenter.bat**.
- Set **dname, alias, keystorePassword** and **truststorePassword** as appropriate, e.g.:

```
set dname="CN=Pine, OU=Beta, O=Gamma, L=Atlanta, C=US"
set alias=pine
set keystorePassword=abc-X-123
set truststorePassword=XYZ-1234
```

- Save the changes.
- Run the batch file **genWebCenter.bat**.
- Change to folder **install_path/ids-j-My-Company-S1/private**.
- Check if the files **webcenter-keystore-pine, webcenter-truststore** and **webcenter-pine.crt** have been created.
- If you open the truststore **webcenter-truststore** with the DirX Identity Manager, you should see the certificate with alias "pine".
- Make sure the keystore's access rights grant Web Center (that is the Tomcat service) read access to the keystore.
- Change to folder **install_path/ssl**.
- Open file **password.properties**.
- Assign the new password to key **truststore**:

```
truststore=XYZ-1234
```

- Save the changes.

10.2.1.2. Configuring Web Center

- Change to Web Center's **WEB-INF** folder.
- Open file **web.xml**.
- Set the value of configuration parameter **com.siemens.webMgr.requestworkflow.keystoreName** to the full path name of the keystore:

  ```
  @DIRXIDENTITY_INST_PATH@/ids-j-My-Company-S1/
  private/webcenter-keystore-pine
  ```

  The expression @DIRXIDENTITY_INST_PATH@ is dynamically replaced at runtime with the DirX Identity installation folder name.

- Set the value of configuration parameter **com.siemens.webMgr.requestworkflow.keyAlias** to "pine".
- Save the changes.
- Open file **password.properties**.
- Assign the keystore password to **webcenterKeystore**:

  ```
  webcenterKeystore=abc-X-123
  ```

- Save the changes.

10.2.2. Another Web Center on a Different Host

Now let's configure single sign-on for a second Web Center instance on another host.

10.2.2.1. Generating a Private Key and a Certificate

On the machine with the Java-based Server:

- Change to folder **install_path/ids-j-My-Company-S1/utils/ssl**.
- Open file **genWebCenter.bat**.
- Set **dname, alias** and **keystorePassword** as appropriate (note that you cannot change the **truststorePassword**), e.g.:

  ```
  set dname="CN=Beech, OU=Beta, O=Gamma, L=Atlanta, C=US"
  set alias=beech
  ```

```
set keystorePassword=Y+9876-c
set truststorePassword=XYZ-1234
```

- Save the changes.
- Run the batch file **genWebCenter.bat**.
- Change to folder **install_path/ids-j-My-Company-S1/private**.
- Check if the files **webcenter-keystore-beech** and **webcenter-beech.crt** have been created.
- Make sure the keystore's access rights grant Web Center (that is the Tomcat service) read access to the keystore.
- If you open the truststore **webcenter-truststore** with the DirX Identity Manager, you should see now two certificates with aliases "beech" and "pine" (from the first sample).

10.2.2.2. Moving the Keystore

Now move (or copy) the keystore **webcenter-keystore-beech** to any folder on the machine hosting the second Web Center instance. Make sure the keystore's access rights grant Web Center (that is the Tomcat service) read access to the keystore.

10.2.2.3. Configuring Web Center

On the machine hosting the second Web Center instance:

- Change to Web Center's **WEB-INF** folder.
- Open file **web.xml**.
- Set the value of configuration parameter **com.siemens.webMgr.requestworkflow.keystoreName** to the full path name of the keystore:

  ```
  ...folder.../webcenter-keystore-pine
  ```

  Replace "...folder..." with the actual folder path name.

- Set the value of configuration parameter **com.siemens.webMgr.requestworkflow.keyAlias** to "beech".
- Save the changes.
- Open file **password.properties**.
- Assign the keystore password to **webcenterKeystore**:

  ```
  webcenterKeystore=Y+9876-c
  ```

- Save the changes.

10.3. Logging

When testing the single sign-on access from Web Center to a Java Server, set the log level for package "com.siemens.idm.server.authenticator" in the Java Server’s Web Admin to "FINEST".

11. Useful Links

Struts

https://struts.apache.org

Struts 1 Tutorials

https://www.roseindia.net/struts

http://courses.coreservlets.com/Course-Materials/struts.html

Open Web Application Security Project (OWASP)

https://www.owasp.org/

OWASP: Securing Tomcat

https://wiki.owasp.org/index.php/Securing_tomcat

W3C: Cross-Origin Resource Sharing

https://www.w3.org/TR/cors

W3C: Web Accessibility Initiative (WAI)

https://www.w3.org/WAI

W3C: Accessible Rich Internet Applications

https://www.w3.org/WAI/intro/aria.php

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.