USM Deployment Guide

Contents

[Introduction 3](#_Toc433890144)

[Intended audience 3](#_Toc433890145)

[Deliverables 3](#_Toc433890146)

[Deployment Stacks 3](#_Toc433890147)

[Prerequisites 4](#_Toc433890148)

[Deployment combinations 5](#_Toc433890149)

[Configuration 5](#_Toc433890150)

[Standalone setup 9](#_Toc433890151)

[Database 9](#_Toc433890152)

[Prerequisites 9](#_Toc433890153)

[Installation 10](#_Toc433890154)

[Datasource 11](#_Toc433890155)

[Installation 11](#_Toc433890156)

[Installation 12](#_Toc433890157)

[Using the admin console 12](#_Toc433890158)

[Manual installation 12](#_Toc433890159)

[Logging 13](#_Toc433890160)

[CAS Authentication configuration for USM Back-end 15](#_Toc433890161)

[Prerequisites 15](#_Toc433890162)

[Configure the USM Back-end for CAS authentication 15](#_Toc433890163)

[Testing CAS-enabled USM Back-end application 19](#_Toc433890164)

[ECAS Authentication Provider Setup for USM Back-end 20](#_Toc433890165)

[Introduction 20](#_Toc433890166)

[Assumptions 20](#_Toc433890167)

[References 20](#_Toc433890168)

[ECAS Authentication Provider Installation Procedure 21](#_Toc433890169)

[WebLogic domain configuration 22](#_Toc433890170)

[Deploy the USM Back-end for ECAS authentication 23](#_Toc433890171)

[Testing CAS-enabled USM Back-end application 23](#_Toc433890172)

# Introduction

## Intended audience

This deployment guide is intended as a guide to the deployment of the USM system. This includes the USM database, backend and web frontend.

## Deliverables

The system’s components are available as a [liquibase](http://www.liquibase.org/) Maven project for the database and a set of WAR files to be deployed to an application server. The war files come in two forms: The default war for each component redirects all traffic to https while the http versions allow connecting over http catering for environments where a reverse proxy handles the secure connection to the internet while internal traffic runs over http. Select the type needed based on your environment.

The liquibase tagged version can be checked out at  
<https://webgate.ec.europa.eu/CITnet/svn/UNIONVMS/tags/USM/USM-1.9.0/database/liquibase/>

The WARs can be downloaded from the nexus maven repository at the following locations:

* HTTPS versions
  + [Authentication-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Authentication-Rest-Service/1.9.0/Authentication-Rest-Service-1.9.0.war) (provides the authentication REST API)
  + [Information-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Information-Rest-Service/1.9.0/Information-Rest-Service-1.9.0.war) (provides the information REST API)
  + [Administration-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Administration-Rest-Service/1.9.0/Administration-Rest-Service-1.9.0.war) (provides the user administration REST API)
  + [administration-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/administration/1.9.0/administration-1.9.0.war) (web front end for user administration)
* HTTP versions
  + [Authentication-http-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Authentication-http-Rest-Service/1.9.0/Authentication-http-Rest-Service-1.9.0.war) (provides the authentication REST API)
  + [Information-http-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Information-http-Rest-Service/1.9.0/Information-http-Rest-Service-1.9.0.war) (provides the information REST API)
  + [Administration-http-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Administration-http-Rest-Service/1.9.0/Administration-http-Rest-Service-1.9.0.war) (provides the user administration REST API)
  + [administration-http-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/administration-http/1.9.0/administration-http-1.9.0.war) (web front end for user administration)
* ECAS versions
  + [Administration-Ecas-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Administration-Ecas-Rest-Service/1.9.0/Administration-Ecas-Rest-Service-1.9.0.war) (ECAS integrated user administration REST API)

For the purpose of Maven based deployments or for further customization via overlays the above artifacts can also be used as dependencies in a POM using eu.europa.ec.mare.usm as the groupId.

## Deployment Stacks

The USM components have been developed and tested to target two deployment stacks:

* Wildfly 8 and Postgres 9
* Weblogic 12 and Oracle 11g

These could be mixed but cross combinations have not been tested.

The frontend is targeted at the latest versions of Chrome, Firefox, Safari as well as Internet Explorer versions 9-11 (as [supported by Angular.js](https://docs.angularjs.org/misc/faq))

## Prerequisites

This Guide assumes the following tools are installed:

* SVN client (eg: Tortoise SVN)
* Java JDK 7 (with the JAVA\_HOME environment variable being set and added to the PATH)
* Maven 3.3.3 (with the M2\_HOME environment variable being set and added to the PATH)

The downloading of the artifacts from nexus or from svn requires a CITnet account with access to the UNIONVMS JIRA project under a developer role.

The maven build of the liquibase project requires the digit’s Nexus repository to be configured in the maven user settings repositories section as well as in the plugin repositories section. The following snippet is an example of this and should be combined with the appropriate server credentials for the ids used.

<profiles>

<profile>

<id>digit</id>

<repositories>

<repository>

<id>citnetnexus</id>

<url>https://webgate.ec.europa.eu/CITnet/nexus/content/groups/public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>citnetplugins</id>

<url>https://webgate.ec.europa.eu/CITnet/nexus/content/groups/public/</url>

<releases>

<enabled>true</enabled>

</releases>

<snapshots>

<enabled>false</enabled>

</snapshots>

</pluginRepository>

</pluginRepositories>

</profile>

</profiles>

This guide also assumes that the chosen application server (AS) / database (DB) is installed and running and that it includes the required database drivers (see confluence documentation for information relating to the setup of the development environment needed for USM).

# Deployment combinations

The aim of the USM system is to provide a user security and management system that other applications can rely on. For the purpose of this, the USM system has been developed to allow different deployment combinations. It may be used fully autonomously, but it may also be used in conjunction with an existing identity management system (IDM) like an LDAP or an active Directory. It may also be combined with ECAS or CAS systems.

The web frontend has been developed as an angular module. The released WAR provides an EC layout implemented as a bootstrap theme. This could be customized and the actual sources for the front end module can be used as a “user management” inside another module or the usm “Auth” module for angular can be re-used independently for other angular based applications.

While this guide covers using USM with an external IDM or ECAS/CAS it will not cover the frontend integration options.

# Terminology

USM uses the following terminology.

User: The user entity in USM represents a person or system with credentials that allow him/it to access the application.

Application: An application is a system using USM for authentication and authorization.

Feature: A feature is an application functionality that can for which access can be controlled.

Dataset: A dataset is an application’s filter on a type of data for which access can be controlled.

Option: An option is an application’s functionality for which each user may have the possibility of having a different setting

Role: A role is a named set of permissions, ie: a group of features that a user may have access to.

Scope: A scope is a named (and optionally time boxed) set of datasets, ie: a group of filters on some data.

User Context: A user context is the combination of a user with a role and an optional scope. A user may have multiple contexts but the User Interface of the application should only allow working within one context at a time.

Policy: A policy is a configurable setting that determines how USM works. It applies to the applications as a whole.

# Configuration

## Bootstrap Data

When installed from scratch the USM database includes a set of data to bootstrap the application. This includes the default policy values as well as the features of the USM application. It also includes a user with a username of **usm\_bootstrap** and a password set to **password**. This user is assigned a role name USM Administrator giving it access to all the features of USM. This user’s password should be changed immediately to prevent security issues.

## Features

The following is the list of features that USM provides.

TODO

## Policies

USM comes with a number of configurable policies. These policies can be listed in the USM user interface under the ***Policies*** menu. A user with the relevant permission can edit policy values. Policy changes should be performed with care as incorrect values might prevent the application from working. It should be noted that some policy changes might take 5 minutes to come into force due to some caching being applied for performance reasons.

A separate guide details the setup and configuration of USM for usage with an external IDM.

### Policy Default Values

|  |  |  |
| --- | --- | --- |
| Policy Name | Description | Default value |
| password.numberOfChallenges | Number of challenges used to amend user information | 3 |
| ldap.enabled | Flag that controls whether LDAP based authentication is enabled (value true) or disabled (value false) | false |
| ldap.server.url | URL for the LDAP server (e.g. ldaps://ldap.domain.org:636/) | ldaps://svm-midway.athens.intrasoft-intl.private:10636/ |
| ldap.context.root | Distinguished Name of the directory node under which users are searched for | ou=users;ou=system |
| ldap.bind.dn | Distinguished Name of the LDAP user account used to query the directory | uid=admin;ou=system |
| ldap.bind.password | Password of the LDAP user account used to query the directory | secret |
| ldap.query.filter | LDAP query used to retrieve the distinguished name of a user, given the USM user name | (&(objectClass=person)(uid={0})) |
| ldap.query.attributes | Comma separated list of LDAP attributes to be retrieved from a user account. The attributed should be listed in the order of first name, last name, phone, mobile, fax and email. If you want to omit some of them then leave the corresponding position empty by ,,. | givenName,sn,telephoneNumber,mobile,facsimileTelephoneNumber,mail |
| ldap.enabled | Flag that controls whether LDAP based authentication is enabled for user administration (value true) or disabled (value false) | true |
| ldap.server.url | URL for the LDAP server for user administration (e.g. ldaps://ldap.domain.org:636/) | ldaps://svm-midway.athens.intrasoft-intl.private:10636/ |
| ldap.context.root | Distinguished Name of the directory node under which users are searched for when using LDAP in user administration | ou=users;ou=system |
| ldap.bind.dn | Distinguished Name of the LDAP user account used to query the directory when using LDAP in user administration | uid=admin;ou=system |
| ldap.bind.password | Password of the LDAP user account used to query the directory | secret |
| ldap.query.filter | LDAP query used to retrieve the distinguished name of a user, given the USM user name when using LDAP in user administration | (&(objectClass=person)(uid={0})) |
| ldap.label.firstName | The user attribute mapping to the firstName label on the UI when using LDAP in user administration | givenName |
| ldap.label.lastName | The user attribute mapping to the lastName label on the UI when using LDAP in user administration | sn |
| ldap.label.telephoneNumber | The user attribute mapping to the telephoneNumber label on the UI when using LDAP in user administration | telephoneNumber |
| ldap.label.mobileNumber | The user attribute mapping to the mobileNumber label on the UI when using LDAP in user administration | mobile |
| ldap.label.faxNumber | The user attribute mapping to the faxNumber label on the UI when using LDAP in user administration | facsimileTelephoneNumber |
| ldap.label.mail | The user attribute mapping to the mail label on the UI when using LDAP in user administration | mail |
| update.contact.details.enabled | Enable (true) or disable (false) the <<Update Contact Details>> feature | true |
| review.contact.details.enabled | Enable (true) or disable (false) the <<Review Contact Details Updates before they become effective>> feature | false |
| account.maxSessionOneSite | Maximum number of concurrent sessions with the same user from a single site | 0 |
| account.maxSessionAnySite | Maximum number of concurrent sessions with the same user from any site | 0 |
| account.maxSessionDuration | Maximum duration of a user session in number of seconds | 28800 |
| account.lockoutFreshold | Number of consecutive failed logins that trigger an account lockout. A value of 0 (zero) disables account lockout for consecutive failed logins | 3 |
| account.lockoutDuration | Account lockout duration in minutes. A value of 0 (zero) disables account lockout for consecutive failed logins | 30 |
| password.renewalReminder | Password renewal reminder at user login in number of days before expiration. A value of 0 (zero) disables reminder at login | 7 |
| password.renewalReminder | Password renewal reminder via e-mail in number of days before expiration. A value of 0 (zero) disables reminder via e-mail | 0 |
| option.valueSize | The size of maximum allowed data for value of an option | 100 |
| password.minLength | Password minimum length | 8 |
| password.minSpecial | Password minimum number of special characters | 0 |
| password.minDigits | Password minimum number of digits | 0 |
| password.minHistory | Minimum number of different passwords before reusing a (previous) password | 1 |
| password.maxValidity | Password validity in days of a freshly changed password. A value of 0 (zero) disables password expiration | 0 |

# 

# Standalone setup

In its simplest configuration the USM system can be used on its own. The following steps allow such a setup. Depending on your combination of AS/DB apply the relevant parts.

## Database

### Prerequisites

USM requires its own schema on your DB. The described setup assumes this schema exists and is named “USM2”.

Liquibase uses a changeset system that allows tracking the state of a database as some form of version control system. This allows it to bring a database to the latest state without having to manually track versions. It does so by installing two tables for this purpose. The data in these tables should not be manipulated manually as it would jeopardize the ability to accurately determine what additional changes have to be applied when updating the database.

It also allows using the same set of sources for different database systems.

This guide will assume an empty database to start with and will not cover version updates.

While liquibase allows using a command line tool to run the database changes, this guide will describe using maven to run liquibase.

Liquibase also allows setting a context for a changeset and USM uses this to allow choosing whether to install the bootstrap data only or to also install a set of testdata.

To facilitate setting the database target and credential it is recommended to update the maven settings.xml file with a profile like the following:

<profile>

<id>mypostgresdb</id>

<properties>

<db.driver>org.postgresql.Driver</db.driver>

<db.url>jdbc:postgresql://someotherserver:5432/usm2</db.url>

<db.user>usm2</db.user>

<db.passwd>mypassword</db.passwd>

</properties>

</profile>

<profile>

For Oracle this profile would look like:

<profile>

<id>maredev</id>

<properties>

<db.driver>oracle.jdbc.driver.OracleDriver</db.driver>

<db.url>jdbc:oracle:thin:@10.240.79.20:1521:maredev</db.url>

<db.user>USM2</db.user>

<db.passwd>password</db.passwd>

</properties>

</profile>

The name of the profile should then be added to the maven command as shown below.

Assuming the liquibase sources have been checked out using SVN from the release tags show in the deliverables section of the document use the following commands running from the root of the liquibase maven project (this is the location of the pom.xml file)

### Installation

In addition to the profile created in the user settings to define the database properties, the profile corresponding to the database type (oracle or postgres) must be activated when installing.

In each command shown below -Ppostgres cab be replaced by -Poracle when running it against an oracle database.

#### Installing the db without test data

mvn liquibase:update -Ppostgres,mypostgresdb

or

mvn liquibase:update -Poracle,maredev

The above installs the database structure and the bootstrap data needed by the system. This includes an admin user as described in the bootstrap data section

#### Installing the db with test data

mvn liquibase:update -Ppostgres,mypostgresdb,testdata

This will also install a set of test data to see some data in the interface.

#### Deleting all the objects from the db

mvn clean -Ppostgres,mypostgresdb

Deleting and installing can be done in a single step using:

mvn clean liquibase:update -Ppostgres,mypostgresdb

#### Generating SQL scripts

Liquibase can also generate an SQL script instead of performing the actual operations. This can be used to generate the SQL statements while running against a local database and passing the resulting script to a database administrator for installation in another server.

mvn liquibase:updateSQL -Ppostgres,mypostgresdb -Dliquibase.migrationSqlOutputFile=C:\path\to\my\USM\_postgres\_script.sql

## Datasource

### Installation

This section describes the datasource setup using the application server admin console.

#### Wildfly

This guide assumes that a management user was created.

Steps to perform:

1. Access the web console at <http://localhost:9990/console> and log in using the management user
2. Go to **Configuration** /**Connector**/**Datasources** tab
3. Click **Add** button
4. Add the datasource attributes

**Name**: jdbc/USM2

**JNDI Name**: java:/jdbc/USM2

and click **Next**

1. Select the JDBC Driver and click **Next**
2. Provide the **Connection URL**, the **Username** the and **Password** for the database
3. Click **Test Configuration**. The server console will notify you if the connection was successful.
4. Click **Done**

#### Weblogic

Steps to perform:

1. Access the web console at <http://localhost:7001/console> and log in using the management user
2. Go to **Services/Data Sources** (left panel)
3. Click **New/Generic Data Source**. Fill in a name and add **jdbc/USM2** as JNDI Name. Click **Next**
4. Choose the **Database Driver** and click **Next.** Click **Next** again.
5. Fill in the **Database Name, Host Name, Port, Database User Name, Password, Confirm Password** fields and click **Next**.
6. In the next screen click **Test Configuration**. The server console will notify you if the connection was successful. Click **Next** and in the next screen choose **AdminServer** in the **Servers** section. This will link the data source to the server.
7. Click **Finish**.

## Installation

Depending on your requirements, download the HTTPS, HTTP or ECAS version of the WAR files (see Deliverables section)

### Using the admin console

This section describes the deployment of the WAR files using the application server admin console.

#### Wildfly

Steps to perform for every WAR file:

1. Access the web console at <http://localhost:9990/console> and log in using the management user
2. Go to **Deployments** tab
3. Click **Add**
4. Select the WAR file and click **Next**
5. Tick the enabled box
6. Click **Save**

#### Weblogic

Steps to perform for every WAR file:

1. Access the web console at <http://localhost:7001/console> and log in using the management user
2. Go to **Deployments** (left panel)
3. Click **Install**
4. Select the WAR file and click **Next**
5. Finish the installation by clicking **Next** and **Finish** and the end.

### Manual installation

This section describes how to deploy manually the WAR files.

#### Wildfly

Steps to perform for every WAR file:

1. Copy it in the $WILDFLY\_HOME/standalone/deployments and unzip into an exploded folder (for example Administration-Rest-Service)
2. Rename the folder adding a **.war** suffix (for example Administration-Rest-Service.war)
3. Create a new empty file using the previous folder name adding a .**dodeploy suffix** (for example Administration-Rest-Service.war.**dodeploy**)

The Wildfly server must be started. If everything is fine, the extension .**dodeploy is changed to .deployed .**

#### Weblogic

TODO

## Logging

The USM system is using Backlog framework for recording the application activity.

This allows reporting and persisting error and warning messages as well as info messages so that the messages can later be retrieved and analyzed.

Every WAR contains a logback.xml file used to configure the logger. The file is located under war\_file/WEB-INF/classes folder.

The xml file contains several appenders, which are components responsible for writing the logging events.

The USM system is using by default RollingFileAppender which appends log events into a file with the capability to rollover the log files. For example, RollingFileAppender can log to a file named log.txt file and, once a certain condition is met, change its logging target to another file.

Every appender has few properties:

1. **file** : the name of the file to write to. If the file does not exist, it is created.

The path must be an absolute path.

The default location is ${WILDFLY\_HOME}\standalone\log\usm\ for Widfly and ${WEBLOGIC\_DOMAIN}\logs\usm\ for Weblogic

1. **rollingPolicy** : it dictates RollingFileAppender's behavior when rollover occurs which involves file moving and renaming.

By default, the USM system is using **TimeBasedRollingPolicy** which defines a rollover policy based on time, for example by day or by month.

It takes one mandatory fileNamePattern property and several optional properties.

The mandatory fileNamePattern property defines the name of the rolled-over (archived) log files.

Its value should consist of the name of the file, plus a suitably placed %d conversion specifier. The %d conversion specifier may contain a date-and-time pattern as specified by the java.text.SimpleDateFormat class. If the date-and-time pattern is omitted, then the default pattern yyyy-MM-dd is assumed.

**The rollover period is inferred from the value of fileNamePattern.**

Please check the LogBack documentation for more details and examples (http://logback.qos.ch/manual/appenders.html ).

Also TimeBasedRollingPolicy supports automatic file compression. This feature is enabled if the value of the fileNamePattern option ends with .gz or .zip.

Another option for the rolling policy is **FixedWindowRollingPolicy which** renames files according to a fixed window algorithm.

Available properties for **FixedWindowRollingPolicy are:**

* + minIndex : an int representing the lower bound for the window's index.
  + maxIndex: an int representing the upper bound for the window's index.
  + fileNamePattern: represents the pattern that will be followed by the FixedWindowRollingPolicy when renaming the log files. It must contain the string %i, which will indicate the position where the value of the current window index will be inserted.

For example, using MyLogFile%i.log associated with minimum and maximum values of 1 and 3 will produce archive files named MyLogFile1.log, MyLogFile2.log and MyLogFile3.log.

Also FixedWindowRollingPolicy supports automatic file compression. This feature is enabled if the value of the fileNamePattern option ends with .gz or .zip.

Sometimes you may wish to archive files essentially by date but at the same time limit the size of each log file, in particular if post-processing tools impose size limits on the log files. In order to address this requirement, logback provides a sub-component for TimeBasedRollingPolicy called SizeAndTimeBasedFNATP.

Example:

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<!-- rollover daily -->

<fileNamePattern>mylog-%d{yyyy-MM-dd}.%i.txt</fileNamePattern>

**<timeBasedFileNamingAndTriggeringPolicy**

**class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">**

**<!-- or whenever the file size reaches 100MB -->**

**<maxFileSize>100MB</maxFileSize>**

**</timeBasedFileNamingAndTriggeringPolicy>**

</rollingPolicy>

Note the "%i" conversion token in addition to "%d". Each time the current log file reaches maxFileSize before the current time period ends, it will be archived with an increasing index, starting at 0.

Loggers may be assigned levels. The set of possible levels is TRACE, DEBUG, INFO, WARN and ERROR.

If a given logger is not assigned a level, then it inherits one from its closest ancestor with an assigned level. To ensure that all loggers can eventually inherit a level, the root logger always has an assigned level.

For the USM system, the root logger level for every war file is specified in logback.xml

<root level="INFO" additivity="false">

If any of these values must be modified (like the log file path, the rolling policy, the logger level), the war file must be unpacked, the logback.xml file must be updated according with the new requirements and the war must be updated.

In other words:

1. copy the war file in new folder
2. extract the war file in this folder
3. modify the WEB-INF/classes/logback.xml file
4. update the war file executing in the folder you just created:

jar -uvf war\_file\_name.war WEB-INF/classes/logback.xml

# CAS Authentication configuration for USM Back-end

This chapter presents the configuration needed to transform USM Back-end to a CAS-enabled application in order to communicate/authenticate using CAS server

## Prerequisites

The CAS server is already installed, configured and available within the premises where the USM back-end application is about to be deployed.

## Configure the USM Back-end for CAS authentication

 Performing this transformation can be achieved by adding CAS profile when building the application for the desired environment. Enabling this profile will transform USM based on the following properties specified in /UNIONVMS/trunk/USM/java/administration/rest/pom.xml file:

1. CAS.casServerUrl  - CAS server URL
2. CAS.serverName – application server host and port where USM Back-end will be deployed. This property can be set by following implicit examples provided in pom file depending on target application server:

* WebLogic: <CAS.serverName>http://${wls.host}:${wls.port}</CAS.serverName>
* JBoss(WildFly): <CAS.serverName>http://${wf.host}:${wf.port}</CAS.serverName>

The following commands can be used to enable CAS on USM back-end:

1. Weblogic: USM\_SRC/java/administration

$ mvn clean install -PCAS,swagger,weblogic -DskipTests

1. JBoss(WildFly): USM\_SRC/java/administration

$ mvn clean install -PCAS,swagger,wildfly -DskipTests

Transforming USM Back-end to a CAS-enabled application consists in two modifications:

1. Adding new filters to web.xml deployment descriptor

                  <!-- The following security constraint enables CAS ${enable.CAS.end}   -->

  <filter>

    <filter-name>CAS Authentication Filter</filter-name>

    <filter-class>org.jasig.cas.client.authentication.AuthenticationFilter</filter-class>

    <init-param>

      <param-name>casServerLoginUrl</param-name>

      <param-value>${CAS.casServerUrl}/login</param-value>

    </init-param>

    <init-param>

      <param-name>serverName</param-name>

      <param-value>${CAS.serverName}</param-value>

    </init-param>

    <init-param>

      <param-name>authenticationRedirectStrategyClass</param-name>

      <param-value>[eu.europa.ec](http://eu.europa.ec).mare.usm.administration.cas.CASAuthenticationRedirectStrategy</param-value>

    </init-param>

  </filter>

  <filter-mapping>

    <filter-name>CAS Authentication Filter</filter-name>

    <url-pattern>/\*</url-pattern>

  </filter-mapping>

  <filter>

    <filter-name>CAS Validation Filter</filter-name>

    <filter-class>org.jasig.cas.client.validation.Cas10TicketValidationFilter</filter-class>

    <init-param>

      <param-name>casServerUrlPrefix</param-name>

      <param-value>${CAS.casServerUrl}</param-value>

    </init-param>

    <init-param>

      <param-name>serverName</param-name>

      <param-value>${CAS.serverName}</param-value>

    </init-param>

  </filter>

  <filter-mapping>

    <filter-name>CAS Validation Filter</filter-name>

    <url-pattern>/\*</url-pattern>

  </filter-mapping>

  <filter>

    <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

    <filter-class>org.jasig.cas.client.util.HttpServletRequestWrapperFilter</filter-class>

  </filter>

  <filter-mapping>

    <filter-name>CAS HttpServletRequest Wrapper Filter</filter-name>

    <url-pattern>/\*</url-pattern>

  </filter-mapping>

   <!--${enable.CAS.start} End of CAS security constraint-->

1. Adding a new custom CASAuthenticationRedirectStrategy (used by CAS authentication filter to redirect when authentication is required). In order to maintain code clean and with minimum dependencies this class is compiled/used only when CAS profile is enabled. This class will extend default redirect strategy of CAS: whenever the request is not authenticated and it is done using X-Requested-With:XMLHttpRequest header the response will be a JSON object { "success" : false, "status" : "ECAS\_AUTHENTICATION\_REQUIRED", "code" : 303, "message" : "session expired" }:

        @Override

**public** **void** redirect(HttpServletRequest request, HttpServletResponse response, String potentialRedirectUrl) **throws** IOException {

**if** ("XMLHttpRequest".equalsIgnoreCase(request.getHeader("X-Requested-With"))) {

                        response.setContentType("application/json");

                        response.setStatus(200);

**final** PrintWriter writer = response.getWriter();

                        writer.write("{ \"success\" : false, \"status\" : \"CAS\_AUTHENTICATION\_REQUIRED\", \"code\" : 303, \"message\" : \"session expired\" }");

                } **else** {

                        response.sendRedirect(potentialRedirectUrl);

                }

        }

        }

Application servers where USM Back-end is deployed need to have in their trust stores certificates of CAS server. There are two possible solutions to achieve this:

1. Import CAS server certificate into JVM trust store used by AS
2. Configure AS to use its own trust store and import CAS server certificate in it

The following command can be used to import a certificate into a trust store (JVM or AS):

$JAVA\_HOME\jre\bin\keytool -import -alias ca -file somecert.cer -keystore cacerts -storepass changeit

Beside this WebLogic needs extra configuration in order to be able to communicate with CAS server (use TLS1 protocol instead of TLS1.2). In order to achieve this following command should be used in Weblogic server start-up script:

1. UNIX $DOMAIN\_HOME/bin/startWebLogic.sh

# START WEBLOGIC

echo "starting weblogic with Java version:"

${JAVA\_HOME}/bin/java ${JAVA\_VM} –version

export ECAS\_OPTIONS="-Dweblogic.wsee.client.ssl.usejdk=true -DUseSunHttpHandler=true -Dhttps.protocols=TLSv1 -Dweblogic.security.SSL.enableJSSE=true -Dweblogic.ssl.JSSEEnabled=true -Dweblogic.security.SSL.protocolVersion=TLS1"

export JAVA\_OPTIONS=$JAVA\_OPTIONS:$ECAS\_OPTIONS

1. Windows %DOMAIN\_HOME%/bin/startWebLogic.cmd

@REM START WEBLOGIC

echo starting weblogic with Java version:

%JAVA\_HOME%\bin\java %JAVA\_VM% -version

SET ECAS\_OPTIONS=-Dweblogic.wsee.client.ssl.usejdk=true -DUseSunHttpHandler=true -Dhttps.protocols=TLSv1 -Dweblogic.security.SSL.enableJSSE=true -Dweblogic.ssl.JSSEEnabled=true -Dweblogic.security.SSL.protocolVersion=TLS1

SET JAVA\_OPTIONS=%JAVA\_OPTIONS% %ECAS\_OPTIONS%

## Testing CAS-enabled USM Back-end application

After deploying CAS-enabled USM back-end application with swagger profile enabled access the following link:

1. JBoss (WildFly): [http:/localhost:8080/usm-administration/](http://http/localhost:8080/usm-administration/)
2. WebLogic: localhost:7001/usm-administration/

When the link is accessed for the first time there should be a redirect to CAS login page. Once the login operation is successful there should be a return link on CAS server to usm-administration page.

# ECAS Authentication Provider Setup for USM Back-end

## Introduction

This chapter presents:

1. how to install the ECAS Authentication Provider
2. how to configure the USM Back-end for ECAS authentication on the basis that:
   * the information provided may be used to restrict access to the USM front-end (more suitable for ECAS authentication since it is targeted to human beings) to end-users authenticated via ECAS.
   * the information provided may be used to restrict access to both the USM front-end and back-end in the case where both components (front and back) are packaged in a single WAR (using the back-end WAR as an overlay for the front-end) or in a single EAR (presumably with shared session configuration)

## Assumptions

As ECAS is an European Commission product that only exists inside the European Commission. It is assumed that the ECAS Authentication Provider and the USM Back-end are to be deployed onto an Oracle Weblogic Application Server instance (primary J2EE platform used by the European Commission).

## References

1. **ECAS Documentation**:

<https://webgate.ec.europa.eu/CITnet/confluence/display/IAM/ECAS>

1. **ECAS Repository**: <https://webgate.ec.europa.eu/CITnet/confluence/display/IAM/ECAS+Forge>
2. **ECAS Client Installation and Configuration Guide – Basic***:*

[Installation and Configuration Guide - Basic.pdf](https://webgate.ec.europa.eu/CITnet/svn/ecas-public/clients/java/tags/4.3.1/doc/ECAS%20Client%20Installation%20and%20Configuration%20Guide%20-%20Basic.pdf)

## ECAS Authentication Provider Installation Procedure

1. Make a backup copy of your Weblogic domain configuration (config.xml and start-sup shell scripts)
2. Download the latest/desired ECAS Client version (currently 4.3.1) from the ECAS Repository (see ). This includes:
   * **ecas-weblogic-10.3-authprovider-4.3.1.jar**: The ECAS Client JAR EcasIdentityAsserterV2 for WebLogic Server 10.3 and above
   * **ecas-demo.ear**: The demo application (optional)
   * **security.properties**: A customized resource bundle to provide descriptions for the EcasIdentityAsserterV2 provider specific page in the WebLogic admin console.
   * **log4j-1.2.15.jar**: a patched log4j JAR
   * **log4j.xml**: The example log4j configuration file (to be adapted to your environment)
3. Download the ECAS Client *Installation and Configuration Guide – Basic* from the ECAS Repository (see ).
4. Install the ECAS Client as described in the *Installation and Configuration Guide – Basic* document. In particular, do make sure that you follow the instructions provided in Section 4.3. *Security requirements* of the aforementioned document to:
   * **Enable strong cryptography**
   * **Disable SSLv3** (see Sample WebLogic start-up script)
   * **Disable SSLv3 for WebLogic-generated Web Service clients** (necessary on Weblogic 12.1.1 but not on version 12.1.2)
5. After completing the installation of the ECAS Client, make sure you take a note of the fact that the Weblogic administration console is now accessible only via the <http://localhost:7001/console/login/LoginForm.jsp> URL (unless your weblogic administrator user is defined in ECAS user database).
6. If your Weblogic server/domain is deployed outside the European Commission, you will most probably integrate with an ECAS Mockup Server instance rather than with the (real) ECAS server instance with which the ECAS Client is (pre-)configured to integrate. If this is the case, you should open the WebLogic administration console (see above), navigate to the ***Security Realms >myrealm >Providers >EcasIdentityAsserterV2*** screen and update the following properties in the ***Provider Specific*** panel (see **Sample WebLogic configuration**:

* **Assurance Level**: LOW (unless you need to accept internal EC users only)
* **Accept Strengths**: BASIC
* **Ecas Base Url**: base URL of your ECAS Server Mockup instance (e.g. <https://svm-midway.athens.intrasoft-intl.private:7012>)
* **Ecas Server Direct Host Name**: host name (or IP address) of your ECAS Server Mockup instance (e.g. *svm-midway.athens.intrasoft-intl.private*)
* **Ecas Server Direct One Way Ssl Port**: port number of your ECAS Server Mockup instance (e.g. *7012*)

1. Restart your Weblogic server instance for the latest changes to take effect.

## WebLogic domain configuration

1. **Sample WebLogic configuration**

The below example displays the suggested configuration for EcasIdentityAsserterV2 in the Weblogic domain configuration file ($DOMAIN\_HOME/config/config.xml).

<sec:authentication-provider xmlns:sch="https://www.cc.cec/cas/schemas" xsi:type="sch:**ecas-identity-asserter-v2Type**">   
<n1:name xmlns:n1="http://www.bea.com/ns/weblogic/90/security">EcasIdentityAsserterV2</n1:name>   
<sch:control-flag>**OPTIONAL**</sch:control-flag>   
<sch:ecas-base-url>[**https://svm-midway.athens.intrasoft-intl.private:7012**](https://svm-midway.athens.intrasoft-intl.private:7012)</sch:ecas-base-url>   
<sch:accept-strengths>**BASIC**</sch:accept-strengths>   
<sch:assurance-level>**LOW**</sch:assurance-level>   
<sch:ecas-server-direct-host-name>**svm-midway.athens.intrasoft-intl.private**</sch:ecas-server-direct-host-name>   
<sch:ecas-server-direct-one-way-ssl-port>**7012**</sch:ecas-server-direct-one-way-ssl-port>   
</sec:authentication-provider>

The below example highlights the mandatory configuration for DefaultAuthenticator in the Weblogic domain configuration file ($DOMAIN\_HOME/config/config.xml).

<sec:authentication-provider xsi:type="wls:**default-authenticatorType**">   
<sec:control-flag>**OPTIONAL**</sec:control-flag>   
<wls:minimum-password-length>5</wls:minimum-password-length>   
</sec:authentication-provider>

1. **Sample WebLogic start-up script**

The below example highlights the mandatory configuration to disable SSLv3, defined in the Weblogic server start-up script ($DOMAIN\_HOME/bin/startWebLogic.sh)

# START WEBLOGIC   
  
echo "starting weblogic with Java version:"   
  
${JAVA\_HOME}/bin/java ${JAVA\_VM} -version   
  
**export ECAS\_OPTIONS="-Dweblogic.wsee.client.ssl.usejdk=true -DUseSunHttpHandler=true -Dhttps.protocols=TLSv1 -Dweblogic.security.SSL.enableJSSE=true -Dweblogic.ssl.JSSEEnabled=true -Dweblogic.security.SSL.protocolVersion=TLS1"**   
**export JAVA\_OPTIONS=$JAVA\_OPTIONS:$ECAS\_OPTIONS**   
  
if [ "${WLS\_REDIRECT\_LOG}" = "" ] ; then   
echo "Starting WLS with line:"   
echo "${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS}"   
${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS}   
else   
echo "Redirecting output from WLS window to ${WLS\_REDIRECT\_LOG}"   
${JAVA\_HOME}/bin/java ${JAVA\_VM} ${MEM\_ARGS} -Dweblogic.Name=${SERVER\_NAME} -Djava.security.policy=${WL\_HOME}/server/lib/weblogic.policy ${JAVA\_OPTIONS} ${PROXY\_SETTINGS} ${SERVER\_CLASS} >"${WLS\_REDIRECT\_LOG}" 2>&1   
fi

## Deploy the USM Back-end for ECAS authentication

In order for the USM system to use the ECAS authentication service, the ECAS version of the administration war ([Administration-Ecas-Rest-Service-1.9.0.war](https://webgate.ec.europa.eu/CITnet/nexus/service/local/repositories/mare/content/eu/europa/ec/mare/usm/Administration-Ecas-Rest-Service/1.7.0/Administration-Ecas-Rest-Service-1.7.0.war)) must be installed (please see Deliverables and Installation sections).

Also the application servers where USM Back-end is deployed need to have in their trust stores certificates of ECAS server. There are two possible solutions to achieve this:

1. Import ECAS server certificate into JVM trust store used by AS
2. Configure AS to use its own trust store and import ECAS server certificate in it

The following command can be used to import a certificate into a trust store (JVM or AS):

$JAVA\_HOME\jre\bin\keytool -import -alias ca -file somecert.cer -keystore cacerts -storepass changeit

## Testing CAS-enabled USM Back-end application

After deploying ECAS-enabled USM back-end application with swagger profile enabled access the following link:

1. WebLogic: <http://localhost:7001/usm-administration/rest/applications/names>

When the link is accessed for the first time there should be a redirect to ECAS login page. Once the login operation is successful there should be a return link on ECAS server to usm-administration page.