SIEMENS

Contents

User Management Component 2.9.2

UMC Federation Adapter Developer Manual

09/2020
A5E50361223-AA

Guidelines

This manual contains notes of varying importance that should be read with care; i.e.:

Important:

Highlights key information on handling the product, the product itself or to a particular part of the documentation.

Note: Provides supplementary information regarding handling the product, the product itself or a specific part of the documentation.

Trademarks

All names identified by ® are registered trademarks of Siemens AG.

The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Security information

Siemens provides products and solutions with industrial security functions that support the secure operation of plants, systems, machines and networks. In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial security concept. Siemens' products and solutions only form one element of such a concept.

Customer is responsible to prevent unauthorized access to its plants, systems, machines and networks. Systems, machines and components should only be connected to the enterprise network or the internet if and to the extent necessary and with appropriate security measures (e.g. use of firewalls and network segmentation) in place.

Additionally, Siemens' guidance on appropriate security measures should be taken into account. For more information about industrial security, please visit https://www.siemens.com/industrialsecurity.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends to apply product updates as soon as available and to always use the latest product versions. Use of product versions that are no longer supported, and failure to apply latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Security RSS Feed under https://www.siemens.com/industrialsecurity.

Siemens AG
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

A5E50361223-AA
20200903_45968

Contents

1 Overview

This manual describes how to extend UMC authentication procedure by integrating an external authentication service within the UMC Identity Provider login procedure. This integration (called "*Federation*") is performed by means of a either built-in or custom module called *adapter*.

Using this feature a user is able to authenticate using several custom authentication methods: an external web IAM (Identity and Access Management) , bio-metric device, RFID reader or specific gesture.

The administrator may also use the possibility to assign an *alias* to every UMC user in order to identify the user during the authentication (for example the serial number of the RFID).

Authentication methods and security level

UMC Identity Provider allows to use multiple built-in *authentication methods*: user + password, PKI, etc. Depending on the method used for authentication the user is allowed to perform or not critical operations: this capability is defined by the *Security Level* associated to the used authentication method.

The possible security levels are:

- **weak**
- **standard**
- **strong**

Some limitations apply to weak or standard authentication, for example, it cannot be used to authenticate on the UMC Web UI.

Like for built-in authentication methods, a security level can be associated also to Federated authentications.

Client and Adapter Registration

For a Federated authentication be accomplished in a secure way, some "registration" procedures can be required, depending on the type of adapter, in order to:

- register a client machine as a trusted machine (UMC Station client, see Machine Roles)
- register the adapter module on the client and/or on the server machine.

Type of Adapters

A UMC adapter can be classified in one of the following three categories:

- **Desktop adapter**: it gets authentication by using hardware or software installed on a desktop client.

- **Web adapter**: it gets authentication by forwarding authentication requests to 3rd party authentication systems (using for example SAML or OIDC). It does not require any installation on the client.
- **Hybrid adapter**: it uses a combination of the previous two to perform a stronger authentication: a token returned by a desktop adapter is verified by a web end-point.

The following table reports a comparison of the different adapter types with respect to strongness and prerequisites:

type	security level	prerequisite	client side registration
Desktop	weak	station client	required
Web	defined during server-side registration	none	none
Hybrid	defined during server-side registration	station client	required

Built-in UMC Adapters

UMC provides some built-in adapters that can be used, after the specific registration and configuration procedure is applied, without the need of developing any custom code.

The built-in adapters actually provided by UMC are listed in the following table, together with the adapter type:

UMC built-in adapter	adapter type
Cookie adapter	Web
Teamcenter RAC	Hybrid
Teamcenter WEB	Web

Cookie Adapter

UMC provides a way to configure authentication based on cookies. A cookie adapter is released with UMC application which allows an external authentication system to integrate with UMC authentication mechanism via cookies. This functionality has been designed to use a third-party IAM together with UMC Web SSO. In this scenario authentication is managed by a "proxy" and forwarded to the UMC adapter using a cookie.

Teamcenter RAC

Integration with Teamcenter Manufacturing allows the users to log into UMC using Teamcenter Manufacturing from Rich Application Client (RAC).

Teamcenter WEB

Integration with Teamcenter Manufacturing via Web allows the users to use Teamcenter Manufacturing to authenticate on UMC.

Custom UMC Adapters

UMC provides a way to fully customize authentication by developing a custom adapter. Desktop, WEB and Hybrid adapters can be developed.

Custom Desktop Adapters

Custom Desktop Adapters can be classified in two different groups:

- Stateful: the adapter knows if a user is logged in and the identity of the logged user. The example of a device for which this adapter can be developed is a card inside a reader. You are logged in until the card is inside; when you remove the card you are logged out.
  A stateful adapter stores the logging status and the user identity inside its p roperties.
- Stateless: the adapter does not store the user identity. Example: a card read by a contactless reader: when you authenticate with the card you are logged in, and you are still logged in when you remove the card.
  Each time a stateless adapter is asked for the user identity, the adapter must retrieve it again, by using an internal mechanism based on events.

UMC provides the possibility to develop both Stateful and Stateless adapters in .NET technology, by developing a concrete class that implements the abstract class provided by UMC.

Custom Web Adapters

Custom Web Adapters can be developed to federate an external IAM (Identity and Access Management) implementing a specific HTTP based protocol. The adapter can be developed using any available technology (ASP.NET, node.js, php, etc) and does not require any client side installation.

1.1 Glossary

Term	Definition
SL UM	SIMATIC Logon User Manager (renamed to UMC)
UMC	User Management Component
UM Ring Server	UMC Server potential master
UM Server	UMC Server
UM Agent	Local agent for authentication
SL IP*	Identity Provider performs Primary Authentication and returns the Claim
User	Individual, who logs on to a computer system and acts in one or more user roles. A user without a role assignment cannot logon to a computer system with a role based access control!
Authentication	Authentication is the process of discovering and verifying the identity of a user by examining the user's credentials and validating those credentials against some authority.
Authorization	Authorization is the process of determining whether a user is allowed to perform a requested action. Authorization generally occurs after authentication, and it uses information about the user's identity to determine what resources the user can access.
Claim	A claim is a statement that one subject, such as a person organization, makes about itself or another subject. The subject making the claim or claims is the provider.
Credential	Data being used to establish the identity of the user, such as the combination of user name and password.
UM Ring	Network of UM Server potential master joined in the same domain
SL Web SSO	web server for Single Sign On (Identity Provider)
UMC-SSO	web server for Single Sign On (Identity Provider)
Federation	specific configuration for the Claim Issuer in order to forward authentication request to a trusted external issuer
Adapter	a software module (either built-in or custom) that is used to integrate an external authentication service within the UMC Identity Provider login procedure
flex authentication	synonymous for Federation
User alias	An alternative name or label that can be assigned to a user in order to identify the user during the authentication
Security Level	Capability assigned to a user to perform or not critical operations, due to the authentication method used to authenticate
umconf	utility that can be used for managing the adapter registration on the UMC ring server

Domain	**A domain is a collection of computers defined by the administrator of a network that share a common directory data-base. A domain provides access to the centralized user accounts and group accounts maintained by the domain administrator.**
RyP	**Relying party**
Provisioning	**Synchronization of the content in the TIA-UM data base with the content in the Windows user data base. The synchronization can be done manual or automatically based on configurable rules.**
IWA	**Integrated Windows Authentication**

2 Concepts You Need to Know About

The following concepts are considered prerequisites to understand UMC Web development:

- Machine Roles
- Deployment Scenarios

2.1 Machine Roles

UMC Computer Roles

In a typical UMC scenario there are three computer roles:

- **UM ring server**: the owner of the UM configuration, which is responsible for managing the domain, and provides full implementation of authentication and user management features. The *priority* ring server is the one which is configured first, running the **umconf** utility. If more than one ring server is available, if you unjoin the priority ring server, the system dynamically elects a new priority ring server.
- **UM server**: provides full implementation of authentication features, the UM server is in *degraded mode* if it is not connected to any UM ring server.
- **UM agent**: works as a client of the UM server/UM ring server to which it is attached, which can be used to run an application developed using the UMC API. See the *User Management Component API SDK Developer Manual* for more details. In order to import Windows Local Users, see **Importing a Windows Local User on an Agent** in the *UMC Installation Manual*.

Important:

Engineering operations are not allowed on the UM Agent except for encryption enablement.

The main differences between the three aforementioned machine roles are listed in the table below.

The ring server to which the other ring servers send the request to write on the UMC database (the candidate for writing) is called *master ring server*. Both the priority and secondary ring server can be master.
If the priority server is master, writing is enabled and the machine can write on the UMC database.

In case of failure, the secondary ring server becomes a master ring server with no writing enabled (*safe mode on*). If the safe mode is switched off using the appropriate umx command, the secondary ring server becomes a master with writing enabled. Consider that some operations on the UMC system configuration are not allowed in this case, e. g. modifying the whitelist (see *UMCONF User Manual* for more details).

UMC Station Client

A machine role orthogonal to the previous ones is *UMC station client*. A UMC station client is a machine where UMC station client software has been installed and that has been registered to be a trusted machine. A UMC station client provides a claim in which certified logon station information are included. These details can be used to associate authorization rights with a machine, which must not be a ring server,server or agent, using the client product.

UMC installation includes UMC station client installation, thus, UM ring servers, UM servers and UM agents need only to register to become UMC station clients, whereas a machine that is not part of the UMC domain has to install the UMC station client software first and then has to register to become a UMC station client.

CAUTION:

If you want to manage Active Directory users, the UM ring server and the UM server machines have to be joined to the AD Windows domain.

Machine Role Functionalities

The table below provides the functionality mapping against the machine roles. For each functionality:

⊕ denotes that the functionality has been fully implemented;

⊖ denotes that the functionality is not available.

⚠ only available when the system is connected to the UM Server

	UM Server	UM Ring Server	UM Agent
Perform TIA User authentication	⊕	⊕	⚠
Local single modifications	⊖	⊕	⊖
Change password	⊕	⊕	⚠
Authentication against Active Directory	⊕	⊕	⊕
Manage Domain attach/join	(acts as proxy for agents) ⊕	⊕	⊖
Potential Master	⊖	⊕	⊖
Can sign authentication object	⊕	⊕	⊖
Propagate UM configuration	⊖	⊕	⊖
Can host Identity Provider /Remote Authentication or UMC Web UI	⊕	⊕	⊖

Number of instances	max 4	1-2	max 25
Off Line Authentication/Read-Only on the configuration	⊕	⊕	⊖
Ring Failover - recovery	⊕	⊕ (merge missing)	⊖
Electronic Log Store&Forward	⊕	⊕	⊖
Log Forwarding	⊖	⊕	⊖
Import Windows Local Users	⊕	⊕	⊕
Import AD Users/Group	⊕	⊕	⊖

2.2 Deployment Scenarios

We support the following deployment scenarios:

- **standalone scenario**: one ring server where UMC and all its Web components are installed and configured: Note that: a quick configuration guide is available for this scenario.
- **redundant scenario**:
  - 2 UM ring server machines, one ring server is configured first and is called *priority ring server*, the secondary one is added to the ring using the join command;
  - up to 4 UM servers
- **distributed scenario**:
  - 1 or 2 UM ring server machines, one ring server is configured first and is called *priority ring server*, the secondary one is added to the ring using the join command;
  - up to 4 UM servers
  - up to 25 UM agents.

Each UMC Web component can be installed and configured on any UM ring server and/or on any UM server. If you install the UMC Web UI on a UM server, you cannot import AD users via UMC Web UI.

NLB redundancy is supported only for Identity Provider.

Standalone engineering station

UMC allows you to prepare configuration data (users, groups and so on) in a standalone engineering station, export this data in a UMC configuration package which can then be imported into a production target system. The two commands involved are the umx export and import package commands. If you want to overwrite the configuration of the target production system with that of the source engineering machine the update command can be used instead of the import command. For more information on these command and how they impact the target machine, see the *UMX User Manual*.

If the target system is not configured, you can import a package using the umconf import package command. For more information see the *UMCONF User Manual*.

3 Installing and Configuring Federation Adapters

In the following sections the specific registration and configuration procedure for built-in adapters and custom adapters is described.

- How to Configure Authentication via Cookie Adapter
- How to Configure Teamcenter Manufacturing Web Integration
- How to Configure Teamcenter Manufacturing RAC Integration
- How to Configure a Custom Desktop Adapter
- How to Configure a Custom Web Adapter

3.1 How to Configure Authentication via Cookie Adapter

UMC provides a way to configure authentication based on cookies. A cookie adapter is released with UMC application which allows an external authentication system to integrate with UMC authentication mechanism via cookies. This functionality has been designed to use a third-party IAM together with UMC Web SSO. In this scenario authentication is managed by a "proxy" and forwarded to the UMC adapter using a cookie.

The adapter is a **umc-cookie-adapter.zip** file that can be found for instance in C:\Program Files\Siemens\UserManagement\WEB\add-ons.

Prerequisites

Three machine roles are involved in the following procedure:

• the *UMC machine* which is a master ring server; a master is a priority ring server or a secondary ring server where safe mode has been disabled and that is promoted to be a master;

• the *external authentication system machine*, where Node.js is installed;

• *the client machine which* is a machine where you have authenticated at least once with the user selected for authentication.

UMC Identity provider must be configured in at least one node of the UMC Network (ring server or server).

In order to protect the cookie adapter, a lockdown procedure must be implemented.

Procedure

1. Copy the file **umc-cookie-adapter.zip** from UMC machine to the external authentication system machine and unzip it.
2. Generate a pair of private/public keys in the external authentication system machine.
3. Register the cookie adapter in the UMC machine using the dedicated **umconf** command; see UMCONF User Manual for more details.

4. List the adapters to retrieve the fingerprint/keyid in the UMC machine using the dedicated umconf command; see *UMCONF User Manual* for more details.
5. Enable login via cookie adapter on the UMC machine, see configuring identity provider in the UMC Installation Manual .
6. Define the user alias for the cookie adapter authentication via Web UI.
7. Edit the file default.json in the external authentication system machine.
8. Edit the script **UMC_Cookie_Federation_Adapter_Start.bat** in the external authentication system machine to set the related local environment variable.
9. Launch **UMC_Cookie_Federation_Adapter_Start.bat** to start the application in the external authentication system machine.
10. Open the Idp login page on a client machine, the cookie adapter is available in the list of the alternative "login options".

Once completed if you select to authenticate via cookie adapter, you are authenticated with the user with whom the alias corresponding to the identity retrieved from the cookie adapter is associated.

Lockdown Procedures

To protect the cookie adapter, a lockdown procedure must be implemented to restrict access to the cookie adapter. Access must be restricted to only allow proxy access. Creating and implementing the lockdown procedure is the responsibility of the application owner. Once you have implemented the lockdown procedure, it is very important that you test it to verify whether it works. This can be accomplished by attempting to access the application by various means, all of which should fail. Your test should be repeated any time you make a change to your system to validate the lockdown procedure has not been removed or modified.

3.1.1 Generating a Private/Public Keys Pair

A pair of private/public keys must be generated which use RSA encoding of at least 2048 bits; we highly recommend that the private key is encoded with an encryption cypher. To do so you can use, for instance, **OpenSSL** tool with the following commands:

- to generate the private key: openssl genrsa -des3 -out private.pem 2048;
- to extract the public key: openssl rsa -in private.pem -outform PEM -pubout -out public.pem.

When generating the private key you will be asked to insert a passphrase, which must be used to set the local environment variable in **UMC_SIGN_PASSPHRASE UMC_Cookie_Federation_Adapter_Start.bat.**

Public key is required during web adapter registration using the appropriate **umconf** command (see UMCONF User Manual for more details).

The fingerprint associated in the public key (sha1) must be set in the appropriate property in the configuration file **default.json**.

3.1.2 Configuring the Cookie Adapter

The file **default.json** is located in umc-cookie-adapter\config.

Configuration JSON example

```
{
  "app": {
     "port": 1987,
     "endPoint": "/cookie-adapter",
     "cookieName": "x-login-user",
     "rejectSelfSSL": false,
     "testPage": "/testpage",
     "debugMode": false,
     "enableHttps": false
   },
  "bouncePage": {
     "formName": "config/umc-cookie-adapter.html",
    "formEncoding": "utf-8"
   },
  "keys": {
     "privateKey": "config/key.pem",
    "certKey": "",
     "cert": ""
   },
  "token": {
     "lifetime": 180,
     "lifetimeUnit": "seconds",
     "jwtHeader": {
       "alg": "RS256",
       "typ": "JWT",
       "kid": "e670121515d18baead7cae858a0d1e8d0ba71db4"
     },
    "jwtPayload": {
       "issuer": "UMC Flex Auth",
       "typ": "PLG",
       "plgId": "3d8203e3-8d5b-48eb-8522-44c30cc33cb0"
     }
  }
}
```

JSON description

The following tables contain the properties for each object.

app Object

The **app** object has the following properties:

Property	Type	Description
port	integer	Port number for listening.

Property	Type	Description
endPoint	string	POST path for retrieving user information.
cookieName	string	Cookie name. ⚠ To be modified.
rejectSelfSSL	boolean	**true** for rejecting self signed SSL, **false** otherwise.
testPage	string	Path for testing GET requests.
debugMode	boolean	**true** for disabling form auto submit, **false** otherwise.
enableHttps	boolean	**true** to enable https between adapter and UMC, **false** otherwise.

bouncePage Object

The **bouncePage** object has the following properties:

Property	Type	Description
formName	string	Relative path of form definition.
formEncoding	string	formName encoding.

keys Object

The **keys** object has the following properties:

Property	Type	Description
privateKey	string	Relative path of private key used to sign json web token. ⚠ To be modified.
certKey	string	Cert key file required by https configuration. To be modified if **enableHttps** is true.
cert	string	cert file with https server certificate. To be modified if **enableHttps** is true.

token Object

The **token** object has the following properties:

Property	Type	Description
lifetime	integer	Session lifetime.
lifetimeUnit	string	Session lifetime unit.
jwtHeader	N/A	See below for the description.
jwtPayload	N/A	See below for the description.

jwtHeader Object

The **jwtHeader** object has the following properties:

Property	Type	Description
alg	string	RS256 by default.
typ	string	JWT by default.
kid	string	Fingerprint of public key. ⚠ To be modified.

jwtPayload Object

The **jwtPayload** object has the following properties:

Property	Type	Description
issuer	string	UMC Flex Auth by default.
typ	string	PLG by default.
plgId	string	the id assigned during registration. ⚠ To be modified.

3.2 How to Configure Teamcenter Manufacturing Web Integration

By using the Teamcenter web adapter, it is possible to implement a Single Sign On between Teamcenter Security Services login page and any product that uses UMC Identity provider as login mechanism.

Integration with Teamcenter Manufacturing via Web allows the users to use Teamcenter Manufacturing to authenticate on UMC. This method, by default configuration, allows strong authentication, see *UMCONF User Manual* for more information on security levels.

The following machines are involved in this configuration and must be configured in this order:

1. Configure the Teamcenter Machine
2. Configure the Web Server
3. Configure the UMC Server

Note: The Web Server described in the following must be reachable by all Clients and the UMC Server.

Configuring the Teamcenter Manufacturing Machine

The following steps must be performed on the machine where Teamcenter Manufacturing is installed and configured.

Prerequisites

Teamcenter 10.1.7 has been installed and configured along with Teamcenter Security Services.

Procedure

1. Retrieve the TC Login Service URL.
2. Specify "UMC" as the **TeamCenter Security Services Application ID.** If you specify an ID which is not "UMC", you must modify the **tcAppId** in the default.json, as described in the following procedure.

Configuring the Web Server for Integration with Teamcenter Manufacturing

The Web server configuration consists in configuring the Web adapter that UMC provides for integration with Teamcenter Manufacturing.

The adapter is a **tcss-web-adapter.zip** file that can be found for instance in C:\Program Files\Siemens\UserManagement\WEB\add-ons.

Prerequisties

NodeJS 10.15.2 has been installed and binary path should be contained in PATH system environments.

Procedure

1. Copy the file **tcss-web-adapter.zip** from UMC machine and unzip it.
2. Retrieve the needed node modules using the command "npm install" inside the **tcss-rac-adapter** folder.
3. Generate a pair of private/public keys.
4. Edit the file default.json.
5. Edit the script **tcss-web-adapter.bat** to set the **UMC_SIGN_PASSPHRASE** local environment variable.
6. Launch **tcss-web-adapter.bat** to start the application.
7. Take note of location of the Public Key file and the URL of the web adapter in as they are required for the next procedure.

Configuring the UMC Server for Integration with Teamcenter Manufacturing

The following steps must be performed on the UMC Server in order to integrate Teamcenter manufacturing with UMC.

Prerequisities

- The TCSS Web Adapter has been launched on the Web Server.
- UMC full installation has been installed and configured.
- The user specified must have the functional right **UM_Admin.**
- If you are using HTTPS a valid SSL certificate must have been acquired from a Certification Authority, or a self-signed SSL certificate must have been created.

Procedure

1. Enable login via " flex authentication" see Identity Provider Configuration in the *UMC Installation Manual*.
2. Run the powershell script **TCSS_Web_Adapter_Server_Configuration.ps1** as administrator, which can be found in the \Wow\BIN under the installation path.
3. Insert the username and password, the url of the adapter, which must be reachable by all the machines which use the adapter, and the location of Public Key. For example:

```
UMC Admin Username: manager
UMC Admin Password: manager
Public Key File full path: C:\Users\Administrator\Desktop\
tcss-web-adapter\config\public.key
endPoint URL (TCSS Web Adapter endPoint): http://web-endpoint:1988/
tcss_web
Plugin registration succesfully
```

4. After running the powershell script, for each machine where the Identity Provider is installed, it is necessary to perform the **Recycle** of the application pool of the Identity Provider (**SimaticLogonPool**, for configuration via script) in **IIS Manager**.

CAUTION:
Performing the **Recycle** of the application pool can cause service interruption.
If the IdP is configured on HTTPS, it may causes issues with browsers other than Internet Explorer, to avoid these issues we recommend to use a reverse proxy.

3.2.1 Generating Private and Public Keys for Teamcenter Web Integration

A pair of private/public keys must be generated which use RSA encoding of at least 2048 bits; we highly recommend that the private key is encoded with an encryption cipher. To do so you can use, for instance, **OpenSSL** tool with the following commands:

- to generate the private key: openssl genrsa -des3 -out private.pem 2048;
- to extract the public key: openssl rsa -in private.pem -outform PEM -pubout -out public.key.

When generating the private key, you will be asked to insert a passphrase, which must be used to set the local environment variable in **UMC_SIGN_PASSPHRASE,** as described in the next steps of How to Configure Teamcenter Manufacturing Web Integration**.**

The fingerprint associated in the public key (sha1) must be set in the appropriate property in the configuration file **default.json**.

3.2.2 Configuring the TCSS Web Adapter

The file **default.json** is located in tcss-web-adapter\config.

The properties which must be specified are:

- **ssoServiceURL**
- **ssoLoginURL**
- **domainName**
- In the **keys** Object:
  – **privateKey**
- in the **jwtHeader** Object
  – **kid**

See the example and tables below for details.

Note: If the IdP has been configured with HTTPS Protocol, and there is no reverse proxy configured, **enableHttps** must be set to true and you must modify the keys Object:

- certKey
- cert

Configuration JSON example

```
{
    "app": {
        "port": 1988,
        "endPoint": "/tcss_web",
        "endPointSetPluginId": "/tcss_web/setpluginid",
        "responseEndPoint": "/tcss_web/getresponse",
        "ssoServiceURL": "http://vmtc101b.swqa.tst:7001/tcssoservice",
        "ssoLoginURL": "http://vmtc101b.swqa.tst:7001/tcssols",
        "tcAppId": "UMC",
        "tcAppUserId": "TCSSO_APP_USER_ID",
        "tcAppSessionKeyToken": "TCSSO_SESSION_KEY",
        "tcDisableApplet": "true",
        "rejectSelfSSL": false,
        "checkPage": "/checkpage",
        "debugMode": false,
        "enableHttps": false
    "domainName" :"mydomain"
    },
    "bouncePage": {
        "formName": "config/umc-tcss-adapter.html",
        "formResponseName": "config/umc-tcss-adapter-response.html"
    },
    "keys": {
        "privateKey": "config/private.key",
        "certKey": "config/certKey.pem",
        "cert": "config/cert.cer"
    },
    "token": {
        "lifetime": 180,
        "lifetimeUnit": "seconds",
        "jwtHeader": {
            "alg": "RS256",
            "typ": "JWT",
            "kid": "88FACEFCD6ED416BC6D516D10E09ABBBDA85FDC6"
        },
        "jwtPayload": {
            "issuer": "UMC Flex Auth",
            "typ": "PLG",
            "plgId": "4c0c2e5c-6e95-4ed1-aa13-591da04c30f8"
        }
    }
}
```

JSON description

The following tables contain the properties for each object.

app Object

The **app** object has the following properties:

Property	Type	Description
port	integer	Port number for listening.
endPoint	string	The relative path of endpoint for the TCSS Web adapter, if this value is modified the values of: responseEndPoint and endPointSetPlugin must also be modified.
endPointSetPlugin	string	The relative path of TCSS Web adapter end point used to automatically register the plug-in id.
responseEndPoint	string	The relative path of the TCSS Web adapter endpoint response.
ssoServiceURL	string	The url of the Teamcenter Security Services identity service. This value must be set by the user.

Property	Type	Description
ssoLoginURL	string	The url of the Teamcenter Security Services login service. This value must be set by the user.
rejectSelfSSL	boolean	**true** to reject self signed SSL, **false** otherwise.
debugMode	boolean	**true** for disabling form auto submit, **false** otherwise.
enableHttps	boolean	**true** to enable https between adapter and UMC, **false** otherwise. Note that if the Identity Provider has been configured with HTTPS this must be set to "true".
DomainName	string	The name of the windows domain which was specified in the Teamcenter Security Service.
tcAppId	string	The application ID which has been configured in Teamcenter Security Service, this value must be set by the user if the ID is not "UMC".
tcAppUserId	string	For Internal use only, the default value is: TCSSO_APP_USER_ID
tcAppSessionKeyToken	string	For internal use only, the default value is: TCSSO_SESSION_KEY
tcDisableApplet	boolean	For internal use only us , the default value is: true
checkPluginId	boolean	For internal use only use, the default value is: true.

keys Object

The **keys** object has the following properties:

Property	Type	Description
privateKey	string	Relative path of private key used to sign json web token. To be modified.
certKey	string	Cert key file required by https configuration. Must be specified if **enableHttps** is true.
cert	string	cert file with https server certificate. Must be specified if **enableHttps** is true.

token Object

The **token** object has the following properties:

Property	Type	Description
lifetime	integer	Session lifetime.
lifetimeUnit	string	Session lifetime unit.
jwtHeader	NA	See below for the description.
jwtPayload	NA	See below for the description.

jwtHeader Object

The **jwtHeader** object has the following properties:

Property	Type	Description
alg	string	RS256 by default.
typ	string	JWT by default.
kid	string	Fingerprint of public key. This value must be set by the user.

jwtPayload Object

The **jwtPayload** object has the following properties:

Property	Type	Description
issuer	string	UMC Flex Auth.
typ	string	PLG.
plgId	string	Plugin id assigned during registration.

3.3 How to Configure Teamcenter Manufacturing RAC Integration

By using the Teamcenter RAC adapter, it is possible to implement a Single Sign On between Teamcenter Rich Client Application (RAC) integrated with Teamcenter Security Services and any product that uses UMC Identity provider as login mechanism.

Integration with Teamcenter Manufacturing allows the users to log into UMC using Teamcenter Manufacturing from Rich Application Client (RAC). This integration is an example of "Hybrid" adapter: a token returned by the desktop adapter installed on the client (TCSS RAC Adapter) is verified by a web end-point (TCSS RAC Web Adapter). This integration method provides therefore strong authentication, see *UMCONF User Manual* for more information on security levels..

The following machines are involved in this configuration and must be configured in this order:

1. Configure the Teamcenter Machine
2. Configure the Web Server
3. Configure the UMC Server
4. Configure UMC Clients

Configuring the Teamcenter Manufacturing Machine

The following steps must be performed on the machine where Teamcenter Manufacturing is installed and configured.

Prerequisites

Teamcenter 10.1.7 has been installed and configured along with Teamcenter Security Services.

Procedure

1. Retrieve the TC Login Service URL.
2. Specify "UMC" as the TeamCenter Security Services Application ID. If you specify an ID which is not "UMC", you must modify the **tcAppId** in the default.json, as described in the following procedure.

Configuring the Web Server

The Web Server configuration consists in configuring the web adapter (TCSS RAC Web Adapter in what follows) that UMC provides for integration with Teamcenter Manufacturing.

The adapter is a **tcss-rac-adapter.zip** file that can be found, for instance, in C:\Program Files\Siemens\UserManagement\WEB\add-ons.

Prerequisties

- NodeJS 10.15.2 has been installed and binary path should be contained in PATH system environments.

Procedure

1. Copy the file **tcss-rac-adapter.zip** from UMC machine and unzip it.
2. Retrieve needed node modules using command "npm install" inside tcss-rac-adapter folder.
3. Generate a pair of private/public keys.
4. Edit the file default.json.
5. Edit the script **tcss-rac-adapter.bat** to set the **UMC_SIGN_PASSPHRASE** local environment variable.
6. Launch **tcss-rac-adapter.bat** to start the application.
7. Take note of location of the Public Key file and the URL of the TCSS RAC Web Adapter as they are required for the next procedure.

Configuring the UMC Server

The following steps must be performed on the UMC Server in order to integrate Teamcenter manufacturing with UMC.

Prerequisities

- UMC full installation has been installed and configured.
- The TCSS RAC Web Adapter has been launched on the Web Server.
- The user specified must have the functional right **UM_Admin**.
- If you are using HTTPS a valid SSL certificate must have been acquired from a Certification Authority, or a self-signed SSL certificate must have been created.

Procedure

1. Enable login via " flex authentication" , see Identity Provider Configuration in the *UMC Installation Manual*.
2. Run the powershell script **TCSS_RAC_Adapter_Server_Configuration.ps1** as administrator, which can be found in the \Wow\BIN under the installation path.
3. Insert username and password, the location of the Public Key file and the URL of the TCSS RAC Web Adapter, which must be reachable by all the machines which use the adapter. For example

```
UMC Admin Username: manager
UMC Admin Password: manager
Public Key File full path: C:\Users\Administrator\Desktop\
tcss-rac-adapter\config\public.key
endPoint URL (TCSS RAC Adapter endPoint): http://rac-endpoint:1987/
tcss_rac
Plugin registration succesfully
```

- After running the powershell script, for each machine where the Identity Provider is installed, it is necessary to perform the **Recycle** of the application pool of the Identity Provider (**SimaticLogonPool**, for configuration via script) in **IIS Manager**.

CAUTION:
Performing the **Recycle** of the application pool can cause service interruption.
If the IdP is configured on HTTPS, it may causes issues with browsers other than Internet Explorer, to avoid these issues we reccomend using a reverse proxy.

Configuring UMC Clients

The following steps must be performed on all the UMC Station Clients, which are the machines where RAC is installed. The powershell script allows you to register a client, if it has not already been registered, and to configure integration with Teamcenter by installing and configuring a desktop adapter (TCSS RAC Adapter).

Prerequisities

- The user specified must have the functional right UM_Admin.
- Teamcenter RAC is installed and configured.
- UMC full or Station Client installation has been installed and the machine has been configured as a Station Client.
- If you are using HTTPS a valid SSL certificate must have been acquired from a Certification Authority or a self-signed SSL certificate has been created.

Procedure

1. Launch the powershell script **TCSS_RAC_Adapter_Client_Configuration.ps1** which can be found in the \Wow\BIN under the installation path.
2. Insert the username, password and the location of the UMC Server. For example:

```
UMC Server (like https://umc-server): http://umc-hostname
UMC Admin Username: manager
UMC Admin Password: manager
Client registration procedure has been started... @{result=success}
Register plugin result:   @{result=success}
```

3.3.1 Generating Private and Public Keys for Teamcenter RAC Integration

A pair of private/public keys must be generated which use RSA encoding of at least 2048 bits; we highly recommend that the private key is encoded with an encryption cipher. To do so you can use, for instance, **OpenSSL** tool with the following commands:

- to generate the private key: openssl genrsa -des3 -out private.pem 2048;
- to extract the public key: openssl rsa -in private.pem -outform PEM -pubout -out public.pem.

When generating the private key, you will be asked to insert a passphrase, which must be used to set the local environment variable in **UMC_SIGN_PASSPHRASE,** as described in the next steps of How to Configure Teamcenter Manufacturing RAC Integration**.**

The fingerprint associated in the public key (sha1) must be set in the appropriate property in the configuration file **default.json**.

3.3.2 Configuring the TCSS RAC Adapter

The file **default.json** is located in tcss-rac-adapter\config.

The properties which must be modified are:

- **ssoServiceURL**
- **ssoLoginURL**
- **domainName**

- In the **keys** Object:
  - **privateKey**
- in the **jwtHeader** Object
  - **kid**

See the example and tables below for details.

Note: If the IdP has been configured with HTTPS Protocol, and there is no reverse proxy configured, **enableHttps** must be set to true and you must modify the keys Object:

- certKey
- cert

Configuration JSON example

```
{
    "app": {
        "port": 1987,
        "endPoint": "/tcss_integration",
        "endPointSetPluginId": "/tcss_integration/setpluginid",
        "rejectSelfSSL": false,
        "checkPage": "/checkpage",
        "debugMode": false,
        "enableHttps": false,
        "ssoServiceURL": "http://vmtc101b.swqa.tst:7001/tcssoservice",
        "ssoLoginURL": "http://vmtc101b.swqa.tst:7001/tcssols",
        "tcAppId": "UMC",
        "tcAppUserId": "TCSSO_APP_USER_ID",
        "tcAppSessionKeyToken": "TCSSO_SESSION_KEY",
        "tcDisableApplet": "true",
        "checkPluginId": true,
    "domainName": "mydomain"
    },
    "bouncePage": {
        "formName": "config/umc-cookie-adapter.html",
        "formEncoding": "utf-8"
    },
    "keys": {
        "privateKey": "config/private.key",
        "certKey": "config/certKey.pem",
        "cert": "config/cert.cer"
    },
    "token": {
        "lifetime": 180,
        "lifetimeUnit": "seconds",
        "jwtHeader": {
            "alg": "RS256",
            "typ": "JWT",
            "kid": "88FACEFCD6ED416BC6D516D10E09ABBBDA85FDC6"
        },
        "jwtPayload": {
            "issuer": "UMC Flex Auth",
            "typ": "PLG",
            "plgId": "b82370af-c3b4-4f54-9007-c34b7233d005"
        }
    }
}
```

JSON description

The following tables contain the properties for each object.

app Object

The **app** object has the following properties:

Property	Type	Description
port	integer	Port number for listening.
endPoint	string	The relative path of the TCSS Rac adapter endpoint, if this value is modified the values of endPointSetPlugin must also be modified**.**
endPointSetPlugin	string	For internal use only, and depends on the value of the previous property.
rejectSelfSSL	boolean	**true** to reject self signed SSL, **false** otherwise.
debugMode	boolean	**true** for disabling form auto submit, **false** otherwise.
enableHttps	boolean	**true** to enable https between adapter and UMC, **false** otherwise. Note that if the Identity Provider has been configured with HTTPS this must be set to "true".
domainName	string	The name of the windows domain which was specified in the Teamcenter Security Service. ⚠ **To be modified.**
ssoServiceURL	string	The url of the Teamcenter Security Services identity service. ⚠ **To be modified.**
ssoLoginURL	string	The url of the Teamcenter Security Services login service. ⚠ **To be modified.**
tcAppId	string	The application ID which has been configured in Teamcenter Security Service.
tcAppUserId	string	For Internal use only, the default value is: TCSSO_APP_USER_ID

Property	Type	Description
tcAppSessionKeyToken	string	For internal use only, the default value is: TCSSO_SESSION_KEY
tcDisableApplet	boolean	For internal use only, the default value is: true
checkPluginId	boolean	For internal use only, the default value is: true.

keys Object

The **keys** object has the following properties:

Property	Type	Description
privateKey	string	Relative path of private key used to sign json web token. ⚠ **To be modified.**
certKey	string	Cert key file required by https configuration. To be modified if **enableHttps** is true.
cert	string	cert file with https server certificate. To be modified if **enableHttps** is true.

token Object

The **token** object has the following properties:

Property	Type	Description
lifetime	integer	Session lifetime.
lifetimeUnit	string	Session lifetime unit.
jwtHeader	NA	See below for the description.
jwtPayload	NA	See below for the description.

jwtHeader Object

The **jwtHeader** object has the following properties:

Property	Type	Description
alg	string	RS256 by default.
typ	string	JWT by default.
kid	string	Fingerprint of public key used during server registration phase. ⚠ **To be modified.**

jwtPayload Object

The **jwtPayload** object has the following properties:

Property	Type	Description
issuer	string	UMC Flex Auth by default.
typ	string	PLG by default.
plgId	string	Plugin ID assigned during registration.

3.4 How to Configure a Custom Desktop Adapter

UMC provides a way to fully customize authentication by developing your own custom desktop adapter. The authentication in this case is *weak.*
For more information on how to develop a custom desktop adapter see Developing Custom Desktop Adapters.

Prerequisites

- the *server machine* that is a master ring server; a master is a priority ring server or a secondary ring server where safe mode has been disabled and that is promoted to be a master;
- the *client machine* is a registered UMC station client.
- a copy of the custom desktop adapter to be installed.

Procedure

1. Register the custom desktop adapter on the server using the dedicated **umconf** command; see *UMCONF User Manual* for more details.
2. Copy the custom desktop adapter to the required client machine.
3. Register the custom desktop adapter on the client machine by using this local endpoint https://localhost:16/slsso/umconf/register_plugin sending adapter information: id, path, classname, type, bitness (please look at Custom Desktop Adapter registration procedure for a detailed information of the registration procedure)
4. Repeat step 2) and 3) for all the client machines that you need to configure
5. Enable login via " flex authentication" see Identity Provider Configuration in the *UMC Installation Manual*.

Important:

UMC Station Client registration and adapter registration can be performed also on any UMC server role.

Result

The custom desktop adapter is listed on the login page when opened on a client machine.

Troubleshooting

If you are not able to see the adapter

1. the client is not register as station client: use the following end point to check the registration status https://localhost:16/slsso/logonstation: if the json returns "result":"success" the registration is ok. If return a different json please register the client and the adapter(s). If an HTTP error is returned the client is not correctly installed: please try to repair the installation.
2. the adapter is not registered on the server: launch umconf -l -P on the server machine and check if the adapter is present in the list (please take note of the id assigned to the adapter)
3. the adapter in not registered on the client : use the following end point https://localhost:16/flexauth/impi/conf and check the returned json: the adapter should be present with the same id of the previous point. If not present try to register again.
4. the adapter is not correctly started: use the following end point https://localhost:16/flexauth/list_impi and check the returned json: the adapter should be present (use the id returned from the previous call to identify it). If not present or the running is false, there is a malfunction on the adapter.
5. the login via " flex authentication" is not enabled. Please check Identity Provider Configuration.

3.5 How to Configure a Custom Web Adapter

Custom Web Adapters can be used to federate an external IAM (Identity and Access Management) implementing a specific HTTP based protocol , and to use this service as authentication provider instead of using UMC Server. For more information on how to develop a custom web adapter see Developing a Custom Web Adapter.

The web adapter does not require client-side installation and works using only redirect/post binding. The final behavior of the SSO will depends on the protocol implemented by the adapter to federate with the external IAM.

By using the keys of the adapter to sign a private key for communications allows the security level of the web adapter to be strong.

Prerequisites

- the *UMC machine* which a master ring server; a master is a priority ring server or a secondary ring server where safe mode has been disabled and that is promoted to be a master;
- UMC Identity provider is configured in at least one node of the UMC Network (ring server or server).
- a copy of the custom web adapter to be installed on the UMC machine.

Workflow

1. Generate a pair of private/public keys in the external authentication system machine.
2. Register the custom web adapter in the UMC machine using the dedicated **UMConf** command; see *UMCONF User Manual* for more details.
3. List the adapters to retrieve the fingerprint/keyid in the UMC machine using the dedicated **UMConf** command; see *UMCONF User Manual* for more details.
4. Enable login via " flex authentication" on the UMC machine, see Identity Provider Configuration in the *UMC Installation Manual* for more details.
5. Ensure that the custom web adapter is active and reachable from your clients.
6. If necessary implement a lockdown procedure.

Result

The custom web adapter is listed on the login page when opened on a client machine.

Troubleshooting

If you are not able to see the adapter

1. the adapter is not registered on the server: launch umconf -l -P on the server machine and check if the adapter is present in the list (please take note of the id assigned to the adapter)
2. the adapter is not correctly started: please check if the adapter is running.
3. the adapter is not reachable from the client: please check the end point of the adapter from a browser window. If you are using https please check the validity of the certificate.
4. the adapter is available only on http: if the Identity provider is configured in https you will run in a "mixed site content" policy violation. Please update adapter configuration.
5. the login via " flex authentication" is not enabled. Please check Identity Provider Configuration.

Lockdown Procedure

To protect the custom web adapter, a lockdown procedure can be implemented to restrict access to the adapter. Access must be restricted to only allow proxy access. Creating and implementing the lockdown procedure is the responsibility of the application owner. Once you have implemented the lockdown procedure, it is very important that you test it to verify whether it works. This can be accomplished by attempting to access the application by various means, all of which should fail. Your test should be repeated any time you make a change to your system to validate the lockdown procedure has not been removed or modified.

3.5.1 Generating a Private/Public Keys Pair

A pair of private/public keys must be generated which use RSA encoding of at least 2048 bits; we highly recommend that the private key is encoded with an encryption cipher. To do so you can use, for instance, **OpenSSL** tool with the following commands:

- to generate the private key: openssl genrsa -des3 -out private.pem 2048;
- to extract the public key: openssl rsa -in private.pem -outform PEM -pubout -out public.pem.

When generating the private key, you will be asked to insert a passphrase, which must be used to set the local environment variable in of the web adapter**.**

Public key is required during web adapter registration using the appropriate **umconf** command (see *UMCONF User Manual* for more details).

The fingerprint associated in the public key (sha1) must be set in the appropriate property in the configuration file **default.json**.

4 Developing Custom Adapters

The following pages provides the details necessary to develop custom adapters:

- Developing Custom Desktop Adapters
- Developing a Custom Web Adapter

4.1 Developing Custom Desktop Adapters

The following pages describe how to develop one of the two types of custom desktop adapters:

- Stateful Adapter, which stores the identity and allows you to retrieve its value using properties,
- Stateless Adapter**,** which does not have a stored identity and retrieve its value during login by user or system action.

Once you have developed your adapter it's necessary to register it: you can also build a script which register it.

4.1.1 Implementing a Stateful Adapter

To develop a custom stateful .Net adapter you have to define a class that implements the abstract class **StatefulPlugin**, which is part of the namespace **PluginDevice**. A .Net dll has to be released in a folder together with:

- **PluginDesign.dll,**
- **um.PluginDesign.dll**

Both dlls can be found in C:\Program Files\Siemens\UserManagement\Wow\BIN.

In what follows we document the abstract method that you have to implement and the two methods that must be used to implement it.

Initialize

This is the abstract method in which you have to implement the initialization logic of the adapter.

```
protected abstract void Initialize();
```

ChangeStatus

This method has to be called once the adapter changes the status.

```
public void ChangeStatus(DeviceStatus deviceStatus)
```

LoadIdentity

This method must be called to load the identity. The string **identity** to be passed as input parameter can be either the name or the user alias set for authentication.

```
protected void LoadIdentity(string identity)
```

Example

The following is a .Net example which returns the Windows identity of the current user.

```
using um.PluginDevice;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Principal;
using System.Text;
using System.Threading.Tasks;
using PluginDevice;
using System.Threading;
namespace WindowsPlugin
{
    public class StatefulWindowsPlugin : StatefulPlugin
    {
        protected override void Initialize()
        {
            this.ChangeStatus(DeviceStatus.Connected);
            var currentWindowsIdentity = WindowsIdentity.GetCurrent();
            this.LoadIdentity(currentWindowsIdentity.User.Value);
        }
    }
}
```

4.1.2 Implementing a Stateless Adapter

To develop a custom Stateless .Net adapter you have to define a class that implements the abstract class **StatelessPlugin**, which is part of the namespace **PluginDevice**.

A.Net dll has to be released in a folder together with:

- **PluginDesign.dll,**
- **um.PluginDesign.dll**

Both dlls can be found in C:\Program Files\Siemens\UserManagement\Wow\BIN.

In what follows we document the two abstract methods that you have to implement and the method that you have to use to implement it.

Initialize

This is the abstract method in which you have to implement the initialization logic for the adapter.

```
protected abstract void Initialize();
```

ChangeStatus

This method must be called once the adapter changes the status.

```
public void ChangeStatus(DeviceStatus deviceStatus)
```

RetrieveIdentity

This is the abstract method in which you have to implement the adapter logic to retrieve the identity. The returned value must match either the user alias set for authentication or the username.

```
protected abstract string RetrieveIdentity()
```

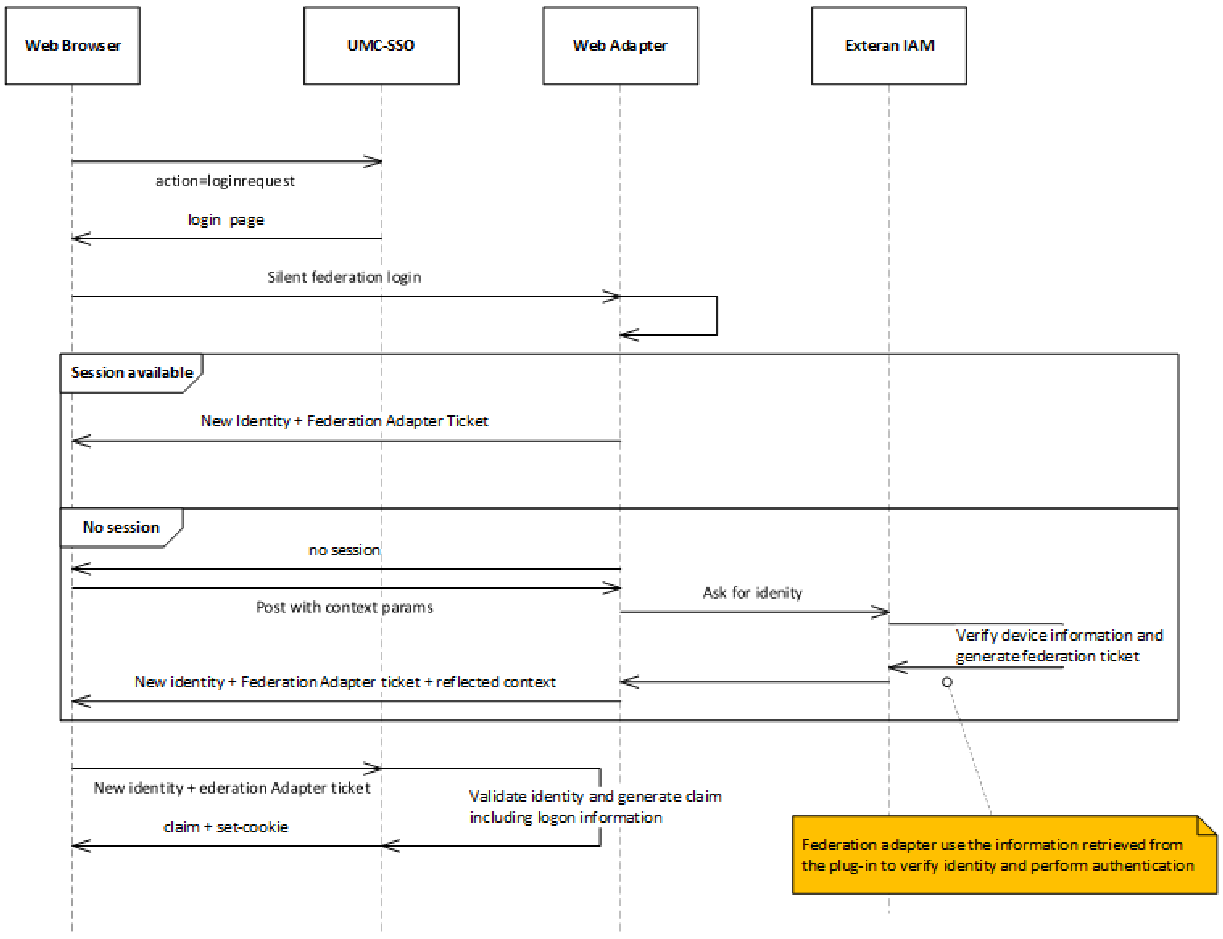
Example

The following example is a .Net example which returns the Windows identity of the current user.

```
using PluginDevice;
using um.PluginDevice;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Principal;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
namespace WindowsPlugin
{
    public class StatelessWindowsPlugin : StatelessPlugin
    {
        protected override void Initialize()
        {
            this.ChangeStatus(DeviceStatus.Connected);
        }
        protected override string RetrieveIdentity()
        {
            var currentWindowsIdentity = WindowsIdentity.GetCurrent();
            return currentWindowsIdentity.User.Value;
        }
    }
}
```

4.2 Developing a Custom Web Adapter

The following diagram describes the flow of data between the browser, IdP, Web Adapter (also called Federation Adapter) and Target IAM. The identity provider starts with a first silent call (using json-cors) and in case of failure (specific error returned from the call) navigates to the end point of the federation adapter.


Web Browser
UMC-SSO
Web Adapter
Exteran IAM
action=loginrequest
login page
Silent federation login
Session available
New Identity + Federation Adapter Ticket
No session
no session
Post with context params
Ask for idenity
Verify device information and generate federation ticket
New identity + Federation Adapter ticket + reflected context
New identity + ederation Adapter ticket
Validate identity and generate claim including logon information
claim + set-cookie
Federation adapter use the information retrieved from the plug-in to verify identity and perform authentication

The federation adapter will start the flow in order to complete the authentication process and will return the result of the authentication to the IdP. The address of the IdP sent in the request is already registered in the adapter and is only used to validate the request. All communication between server and clients is performed using the HTTPS protocol and POST.

The following documentation describes the basic requirements to develop a custom federation web adapter that allows an external authentication system to integrate with UMC authentication mechanism, via HTTP POST requests. This functionality has been designed to use a third-party IAM together with UMC Web SSO.

Required Message Exchange

As described in the flow diagram on a series of POST requests are exchanged of which only the following messages must be handled, either by a listener or by providing the required information, by your adapter:

- An Initial Browser AJAX POST Request (Message 1)
- Web Adapter Response JSON-CORS Message (Message 2)
- IdP POST Request (Message 3)
- Web Adapter POST message sent by the web adapter to the IdP containing the UMC Ticket. (Message 4)

Initial AJAX POST Request (Message 1)

The **browser** sends an initial AJAX request of type POST to the adapter with the following parameters:

```
UMCSSOJson: yes
nonce: 831b316b-40a3-48f3-bbf8-b47679ec2da4__b3mfiyd2rcae5tzfrdm1eumi
```

Field	Description
UMCSSOJson:	If yes then initial message exchange is via JSON-CORS (Silent).
Nonce	The nonce is used to identify that the session registration is related to a previous request of flex authentication.

Response JSON-CORS Message (Message 2)

The **adapter** then must send a message which specifies whether a session is already open with the external IAM or not and replies with a JSON-CORS message containing the parameters described, in the case of silent login the message exchange ends at this step if the identity is successfully retrieved.

Field	Description
UMCTicket	UMC Ticket
UMCResult	Can either be "success" which means there is already a valid session or "no_identity" if there's no session.

Field	Description
UMCUser	The logged user.
UMCSSOLanguage	The language selected by the user. For future use.
AdapterInfo	The Xframe options which must be specified by the adapter if the federated IAM service has Xframe settings which are not compatible with SWAC component. This section is encoded in base64.

```
{"UMCTicket":"","UMCResult":"no_identity","UMCUser":"","UMCSSOLanguage":"",
"AdapterInfo":"eyJYRnJhbWVPcHRpb25zIjoiIn0="}
```

IdP HTTP POST (Message 3)

The **browser** sends a message via HTTPS POST to the adapter with the following fields which are provided by the Identity Provider:

Fields	Description
address	Address of the federation adapter (as configured in UMC).
method	POST
parameters	• UMCReturnAddress: the endpoint used to navigate back to the IdP. • UMCSSOLanguage: the language selected by the user. • json: [optional] If the parameter value is “json” the request is silent, otherwise the request is not silent. • UMCSSO... : Any other parameters which are to be reflected.

Web Adapter POST(Message 4)

Once the request is completed the **adapter** must auto-submit a form containing the following parameters:

Fields	Description
UMCResult	The result of authentication request.
UMCUser	The logged user.
UMCTicket	The flexauth ticket to authenticate.
UMCSSOLanguage	The returned language.
UMCSSO...	The reflected parameters.
USERInfo	For Future use. Additional information on the user, which is specified by the IAM, that will be included in the claim.

The adapter must be able to handle the request and, after the challenge with the external IAM, answer sending a post request containing the information described above.

The following is an example and does not contain a valid UMC ticket see UMC Ticket Details for an example ticket and more information. In the example also the id attribute of the input form fields are filled, but are not mandatory.

Example

```
<form name="myForm" action="https://myreturn" method="post">
         <fieldset>
            <input type="submit" value="Login">
  <input type="hidden" name="UMCSSOUserName" id="UMCSSOUserName" value=
"4f619de2-0bac-4c2c-ae1b-c233de595f5f__ewyxuhn2rtfe13cghjrhapel">
  <input type="hidden" name="UMCSSOTicket" id="UMCSSOTicket" value="">
  <input type="hidden" name="UMCSSOAuthenticationType" id=
"UMCSSOAuthenticationType" value="flexauth">
  <input type="hidden" name="UMCSSOSourceName" id="UMCSSOSourceName" value=
"Login"><input type="hidden" name="UMCSSOSelectedCulture"     id=
"UMCSSOSelectedCulture" value="it-IT">
  <input type="hidden" name="UMCSSOIPReqId" id="UMCSSOIPReqId" value=
"Cgl7CgkJImFjdGlvbiIgOiAibG9naW5yZXF1ZXN0I
      iwKCQkic2VydmljZSIgOiAiaHR0cHM6Ly92bS1pYy0w
      MS9ibHVlL1Jlc3BvbnNlZm9ybS5hc3B4IiwKCQkiY3R4IiA6ICI
      yMDE4LTAzLTE1VDE1OjI4OjQ5WiIsCgkJIm5vbmNlIiA6ICI0ZjYxOWRlMi0wYmFjLTRj
      MmMtYWUxYi1jMjMzZGU1OTVmNWZfX2V3eXh1aG4ycnRmZ
      TEzY2doanJoYXBlbCIsCgkJInNpbGVudCIgOiAibm8iCgl9
      rZaRwmdvg1e0L1a4uFpNxlHpxLdtZpz9ZxsF%2FOvATsXleZRxB6YC8T3iT23CPx
      B3GAeEtZi4AO%2FgxC2e%2F3KML5KaPkhv8mWgWJJotirONUcQQnou5pLkZooLZ0

9i7X%2B5SRgT%2FsL3XZP0X7CM4RNVJA2AZ%2BFGPe7uITwVYd3RRnaCt%2BnbFZWKlStm%2BiD
      tnklxRLnVBcwI5p%2Fs%2Fnoex0QSFIBWb5KuVQOUBMl11LbjbmXHH9kqkEj%
      2FKd8rOzQo3Y8axpTZzjN2WLCbw6aGtdwhiBgdcJTS4Xsfm6%2BXW9yUr%2FEc6taly
      jhhDk%2BsheaRZWM%2FKbhSbeMqu0jyj6MmeeYYoA%3D%3D">
  <input type="hidden" name="UMCSSOComputerTicket"
    id="UMCSSOComputerTicket"
    value="eyJhbGciOiJSUzI1NiIsImtpZCI6IjFEMkEwQTd
    BQkNBNzBBMkYwOUZEQ0UwNThEM0MyQjk4MUJBODFBMzIiLCJ0eXAiOiJKV1QifQ
    ==.eyJleHAiOjEzMTY1NTk3OTE0ODg4NTgyOCwiaWF0Ijox

MzE2NTU5NzczNDg4ODU4MjgsImlzcyI6InZtLWljLTAxIiwic3ViIjoiVk0tVkRJUDEwLTExIn
    0=.ftTVKCrzfMrQcM8gQaoq7zY7TpwZ79O3OXwjqXKVakvge5F0G2PWkkqR/
vF4nOldNLutgkY12J77OFvNszjjsNP/AzAS4Rrpy2aA
    /ftXd4i5GjdtWSVRrtR1SecFO3XgwhKq5LD4PMlmJqwNmyYQVJgr0faoZNL6KfwCQT5NMKM=
">
<input type="hidden" name="UMCResult" id="UMCResult" value="success"><input
type="hidden" name="UMCUser" id="UMCUser" value=""><input type="hidden" name=
"UMCSSOLanguage" id="UMCSSOLanguage" value="">
  <input type="hidden" name="AdapterInfo" id="AdapterInfo" value=
"eyJYRnJhbWVPcHRpb25zIjoiIn0=">
  <input type="hidden" name="UMCTicket" id="UMCTicket"
    value=
"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6IjQ4QkIzNEQxRDEwRDM2N0R
```

```
    EQkU3NDEyQUM0QkQ5OTJBMTZBNjQ1RDMifQ==.eyJleHAiOiIyMDE4LTAzLTE
    1VDE0OjMyOjIxLjAxMVoiLCJpYXQiOiIyMDE4LTAzLTE1VDE0OjI5OjIxLjAxMVoiLCJpc3Mi
    OiJVTUMgRmxleCBBdXRoIiwic3ViIjoiZXlKMGVYQWlPaUpRVEVjaWZRPT0uZXlK
    cFpHVnVkR2wwZVNJNklrRmtiV2x1YVhOMGNtRjBiM0lpTENKd2JIVm5hVzVKWkNJNklqZzVO
    amRoWXprMExUQTRZVFF0TkdVME9TMDVZMlZpTFdNd1lqVm1PRE0yTVdJeVpDSXNJbkJzZF
    dkcGJsTnBaMjVoZEhWeVpTSTZJaUo5In0=.
WjWURvq9Xv4w2WX4U64rcMwJ9SOpxkd5DOBc7jzwYghm
    DZLnZ+5OKwdhCzv3+KENaF+d9/KPA3y1AWN4ckYFhiPYEUG8pS/L9KUbQxSLDgPS
    tQnxaRvL8jDCLNqIlt5FYJzlWoXh9EKx9b1bS2IUERjtT94Gqo06GjyBfVBQV/
yBsAnqWN5PeuIrxPCOhmSw7+
    p0AxkzeoEL4ETJdoJPA+MnIXgVJM72r1MbjjDY/U3anJXU8R3RqaTxUPBRKbd
    WNN9NApEjAJzAjQTXqO/R/bjvzsD7bfetZq0XDtKVnIWUui8dU/
fD5WiHDdakx2V6bq4CP0qHqlKdELO9SzOdvg==">
        </fieldset>
      </form>
```

4.2.1 UMC Ticket

The ticket which must be generated by the web adapter during the authentication process and is composed of a json web token which is made up of header, payload and signature (separated by '.' character).

Header

The header must be constructed as follows:

Key	Value
alg	The fixed value "RS256".
type	The fixed value "JWT".
kid	The key id used during server registration procedure of the adapter (can be also retrieved using UMConf).

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "48BB34D1D10D367DDBE7412AC4BD992A16A645D3"
}
```

Payload

The payload must be constructed as follows:

Key	Value
exp	The expiry date of the ticket.

Key	Value
iat	The issue time of the ticket.
iss	The fixed value which is the issuer of the ticket, "UMC Flex Auth".
sub	See below.

```
{
  "exp": "2018-03-20T15:16:26.101Z",
  "iat": "2018-03-20T15:13:26.101Z",
  "iss": "UMC Flex Auth",
  "sub": "eyJ0eXAiOiJQTEcifQ==.
eyJpZGVudGl0eSI6IkFkbWluaXN0cmF0b3IiLCJwbHVnaW5JZCI6Ijg5NjdhYzk0LTA4YTQt
NGU0OS05Y2ViLWMwYjVmODM2MWIyZCIsInBsdWdpblNpZ25hdHVyZSI6IiJ9"
}
```

Sub

Sub is composed by 2 sub strings encoded base 64 and concatenated using the '.' character:

Key	Value
typ	Fixed value "PLG".

Identity

Key	Value
identity	The user name or the alias of the authenticated user, depending on how the adapter has been configured.
pluginId	The ID which is assigned to the adapter during registration via UMConf.
pluginSignature	N/A

```
//first part
{
  "typ": "PLG"
}


//second part
{
  "identity": "Administrator",
  "pluginId": "8967ac94-08a4-4e49-9ceb-c0b5f8361b2d",
  "pluginSignature": ""
}
```

Signature

The signature contains the signature of the string obtained concatenating the 2 payloads (encoded BASE64) using the '.' character using the adapter private keys and SHA256 RSA algorithm.

Ticket Example

The following is an example of code realized using node.js that illustrates the structure of a UMC Ticket.

Example UMC Ticket

```
    var jwtPayload_identity = identity;


    // private key
    var pair = {
        private: fs.readFileSync(path.join(__dirname, privateKeyFilename),
'utf8')
    };

    //Header
    var jwtHeader = {
        "alg": jwtHeader_alg,
        "typ": jwtHeader_typ,
        "kid": jwtHeader_kid
    }

    //Payload
    var jwtPayloadSubFirst = {
        "typ": jwtPayload_typ
    }

    var jwtPayloadSubLast = {
        "identity": jwtPayload_identity,
        "pluginId": jwtPayload_pluginId,
        "pluginSignature": ""
    }

    var base64_jwtPayloadSubFirst = new Buffer(JSON.stringify
(jwtPayloadSubFirst)).toString("base64");
    var base64_jwtPayloadSubLast = new Buffer(JSON.stringify
(jwtPayloadSubLast)).toString("base64");

    var exp = moment().add(durationBeforeExpire, durationBeforeExpireUnit).
valueOf();
    var expTimestamp = new Date(exp);
    var expFormatted = expTimestamp.toISOString();

    var iat = moment().valueOf();
    var iatTimestamp = new Date(iat);
    var iatFormatted = iatTimestamp.toISOString();

    var jwtPayload = {
        "exp": expFormatted,
        "iat": iatFormatted,
        "iss": jwtPayload_issuer,
        "sub": base64_jwtPayloadSubFirst + '.' + base64_jwtPayloadSubLast
    }

    var base64_jwtHeader = new Buffer(JSON.stringify(jwtHeader)).toString
("base64");
    var base64_jwtPayload = new Buffer(JSON.stringify(jwtPayload)).toString
("base64");

    //Signature
    const crypto = require('crypto');
    const sign = crypto.createSign('RSA-SHA256');

    sign.update(base64_jwtHeader + '.' + base64_jwtPayload);

    return base64_jwtHeader + '.' + base64_jwtPayload + '.' + sign.sign({
key: pair.private, passphrase:process.env.UMC_SIGN_PASSPHRASE } , 'base64');
}
```

5 Appendix

5.1 Custom Desktop Adapter registration procedure

In order to use a desktop adapter for authentication it's necessary to register it not only on the UMC server but also on the client machine: during the registration procedure a digital signature of the binary is performed to improve the validation when the adapter is launched.

To register the adapter it's necessary to call the following end point (POST)

https://localhost:16/slsso/umconf/register_plugin

request body

```
{

'piuid' = '[adapter UID]'

'path' = '[adapter path]'
'name' = '[adapter class name]'
'desc' = '[not used - Description] '
'type' = '[adapter type (stateful/stateless)]'
'adapter' = '.net'
'bitness' = '[adapter bitness (x86/x64)]' 'dsso' = true/false

}
```

where:

[adapter UID] → the id of the adapter (returned during the server registration procedure).

[adapter path] → the full path of the adapter

[adapter class name] → the class name of the object to be instantiate in the .Net adapter

[adapter type (stateful/stateless)] → if the adapter is statefull or stateless

[adapter bitness (x86/x64)] → bitness of the adapter

[dsso] → dSSO Service Integration (boolean value)

```
ContentType

'application/json'
```

Security

The request uses the Integrated Windows Authentication to protect the operation. Only user in the built-in *Administrators* group can perform the registration. Unauthenticated request will be refused.

It's possible to build a script to automate the registration on all the clients (see Building a Script for Adapter Activation).

5.1.1 Building a Script for Adapter Activation

In what follows we provide an example of a PowerShell script to register an adapter. The adapter id can be retrieved by using the dedicated **umconf** command to list the registered adapter on the server. See *UMCONF User Manual* for more details.

```
$plugin=@{
 piuid = Read-Host -prompt '[adapter UID]'
 path = Read-Host -prompt '[adapter path]'
 name = Read-Host -prompt '[adapter class name]'
 desc= 'not used'
 type = Read-Host -prompt '[adapter type (stateful/stateless) ]'
 adapter = '.net'
 bitness = Read-Host -prompt '[adapter bitness (x86/x64)]'
 dsso = try{[bool][int](Read-Host -Prompt '[dsso integration (1, 0)]')}
catch { $false }
 }

$json = $plugin | ConvertTo-Json

$response = Invoke-RestMethod 'https://localhost:16/slsso/umconf/
register_plugin' -Method post -UseDefaultCredentials -Body $json
-ContentType 'application/json'

write-host $response
```

5.2 Setting User Alias for Federation

A *user alias* can be assigned to every UMC user in order to identify the user during the authentication (for ex. the serial number of the RFID).

The Federation authentication mechanism matches the identity provided by the adapter to the user alias or the username stored in UMC.

Depending on the adapter used two different methods can be used to provide the alias:

- **Stateful** adapter uses the **LoadIdentity** method
- **Stateless** adapter uses the **RetrieveIdentity** method

To configure this data matching using the UMC Web UI, you can define an alias for a user in the account policy tab of the user detail dialog; the value stored in the field is compared with the loaded identity and if they correspond the user is authenticated. For more information see the *User Management Component Web User Interface Manual*.

To define an alias you can also use the dedicated UMX command. See *UMX User Manual* for more details.

Example

Consider a user with the following value for the user alias:

Alias = john.brown@mycompany.com

The call of the **LoadIdentity** method would be: **LoadIdentity("john.brown@mycompany.com")**.