Ubisense Installation Guide

for SmartSpace Web

Version 3.8 SP5

Part Number: WEB_INS_3.8 SP5_EN

Ubisense Limited, St Andrew's House, St Andrew's Road, Cambridge CB4 1DL, United Kingdom.

Telephone: +44 (0)1223 535170. Website: *www.ubisense.com*.

Ubisense Limited
St Andrew's House
St Andrew's Road
Cambridge CB4 1DL
United Kingdom

Tel: +44 (0)1223 535170

WWW: *https://www.ubisense.com*

Contents

Installing SmartSpace Web

This guide describes installing and configuring the web server required by the features in the Visibility and Reporting components of SmartSpace, and the Application REST API.

SmartSpace Web and the SmartSpace Application REST API can be deployed on either Windows or Linux web servers. See your installation guide for information on server software supported by Ubisense.

- Under Windows, the websites are installed using Windows Installer, and are controlled and hosted under IIS (described in *Installing SmartSpace Web on Windows*).
- On Linux, the websites are controlled by the Ubisense platform controller, as services, and by default require a reverse proxy (such as Apache2) to make them available to the network (see *Installing SmartSpace Web on Linux*).

You must configure a separate web server for the Application REST API.

These instructions make the following assumptions:

- You have read and followed the installation instructions for your combination of Ubisense products, for example SmartSpace only, or SmartSpace plus another product such as DIMENSION4
- You have installed *as a minimum* the SmartSpace Core features
- You have installed any additional features you have licensed, including those requiring a web server
- *For Windows servers*, you have access to the installation files in your distribution

If you are using unicast cluster support in your Ubisense platform, see also *Visibility and cluster mode* in the *SmartSpace Unicast Cluster Setup Guide* for additional configuration requirements.

After you have configured your web server, you can then use the information in the guides to your web-based features to perform any feature-specific configuration that's required.

For larger installations, additional information on scaling out the SmartSpace website is given in *Scaling out SmartSpace Web*.

Requirements

SmartSpace Web installations have the following requirements. They may be additional to those described in your SmartSpace installation guide.

Microsoft .NET Runtimes

Some SmartSpace features use Microsoft .NET runtimes. These are:

- Reporting
- External data connector
- Typed API
- Real-time rules engine
- SmartSpace Web site, includes:
    - Web maps
    - Web forms
    - HMIs
    - ObjectView API
    - Operations web interface
- Application REST API
- .NET API
- Application .NET API (Managed Browser)
- AVL/GPS connect
- the translation tool used in localization and tailoring

Note: On Windows, it is important that you enable and configure IIS ***before*** you install Microsoft .NET in order that IIS picks up the .NET Runtimes.

For links to the requisite downloads, see *https://dotnet.microsoft.com/en-us/download/dotnet/6.0*.

Current versions of .NET for SmartSpace are:

- For Windows, you *must* install the latest ASP.NET Core Runtime 6.0.x Hosting bundle.
- For Linux, we *strongly recommend* you install the latest ASP.NET Core Runtime 6.0.x.

For a list of SmartSpace releases and their respective .NET versions, see the Ubisense Documentation Portal.

Database Server for Reporting

Either of the following are required *only* if the Reporting component is licensed:

SQL Server

Database versions: 2012 or higher.

- Windows servers using ODBC and ODBC Driver for SQL Server 18.x

  (See *System requirements, installation, and driver files* for details of the driver required for your database version)

- Linux servers using Microsoft ODBC Driver 17 for SQL Server

Oracle

Database versions: 11G R2 or higher.

- Windows servers using Oracle Instant Client 21.x library
- Linux servers using Oracle Instant Client 21.x library

For information on configuring servers for use with the Reporting component, see SmartSpace Reporting on the Ubisense Documentation Portal.

Supported Browsers

Supported browsers for use with SmartSpace's web-enabled features are recent versions of:

- Microsoft Edge
- Chrome
- Chrome for Android
- iOS Safari

Additionally, recent versions of the following browsers work but are not explicitly supported:

- Firefox
- Opera
- Safari

Internet Explorer 11 is deprecated and SmartSpace's web-enabled features are not guaranteed to work with this browser.

Installing SmartSpace Web on Windows

If you have licensed SmartSpace features that are accessed in a browser, such as Web maps or Web forms, you need to set up a web server before installing these features.

To install and configure the Web Server:

1. Enable Internet Information Services.
2. Install the Windows ASP.NET Core Runtime 6.0.x Hosting bundle.
3. Install the websites required.
4. Modify the website configuration files, if required.

Note: On Windows, it is important that you enable and configure IIS ***before*** you install Microsoft .NET in order that IIS picks up the .NET Runtimes.

Enable Internet Information Services (IIS)

Enable Internet Information Services (IIS) on Windows Server 2016

1. Open the Server Manager, click Add roles and features.
2. Click through to Server Roles, and select Web Server (IIS).
3. Click through to Web Server Role (IIS) and under Role Services, ensure that, in addition to the default features, you have enabled Security/Windows Authentication.
4. Click through to Confirm the installation.

Enable Internet Information Services (IIS) on Windows 10 or Windows 11

1. From the Start menu, choose Turn Windows features on or off.
2. In the Windows Features dialog, select Internet Information Services.
3. Expand Internet Information Services/World Wide Web Services/Security and ensure Windows Authentication is selected.
4. Click OK to confirm the installation.

Due its increased vulnerabilities, NTLM is no longer added as a provider for Windows authentication, when SmartSpace Web is installed. If you need to use NTLM, you can manually add it as a provider in IIS Manager.
Firefox does not support Windows authentication by default without NTLM. If you need to support Firefox, you can either add NTLM manually or use this workaround:
The workaround for Firefox requires the browser settings to be changed:

1. Open Firefox and enter **about:config** in the address bar. Click to confirm advanced configuration.
2. In the Filter field, enter **negotiate**.
3. Double-click **network.negotiate-auth.trusted-uris**. This preference lists the trusted sites for Kerberos authentication.
4. Enter your domain (e.g. **company.local**)
5. Click Save (the tick button).

Install the Windows ASP.NET Core Runtime Hosting Bundle

1. Download the Microsoft ASP.NET Core Runtime 6.0.x Hosting bundle from *https://dotnet.microsoft.com/en-us/download/dotnet/6.0* which includes the .NET Runtime and IIS support.
2. Run the installer.
3. If .NET was not previously installed on the server, then a reboot is required for IIS to pick up the path to the .NET Runtime.

Install the Website and/or REST API

Follow these instructions to install the SmartSpace Web application.

When you install the SmartSpace Web application, the following components are created as part of the installation process:

Component	Description
Application Pool:	SmartSpace has its own application pool.
Website:	The entry point to SmartSpace via a browser.

To install the SmartSpace Web application:

1. Go to the **web\windows** directory of your SmartSpace distribution directory.
2. Double-click the **SmartSpaceWeb.msi**.
3. Enter a Website Name: this name will form part of the URL when accessing SmartSpace in a browser.
4. Choose the location for the software.

   You can accept the default **C:\Program Files (x86)\Ubisense 3\SmartSpace** or click **Change** to select another destination.
5. Enter an Application Pool name.
6. Click **Next** and **Install**.

To install the SmartSpace Web API:

1. Go to the **web\windows** directory of your SmartSpace distribution directory.
2. Double-click the **SmartSpaceWebApi.msi**.
3. Enter a Website Name: this name will form part of the URL when accessing SmartSpace in a browser.
4. Choose the location for the software.

   You can accept the default **C:\Program Files (x86)\Ubisense 3\WebApiCore** or click **Change** to select another destination.
5. Enter an Application Pool name.
6. Click **Next** and **Install**.

By default, the SmartSpace website can be accessed by navigating to **http://localhost/smartspace** and the REST API can be accessed by navigating to **http://localhost/smartspaceapi**.

Workaround for Failed Installation due to PowerShell Configuration

Installation of SmartSpace Web or the Application REST API can sometimes fail and rollback. This can be because the local machine has been configured to require signed PowerShell scripts. The workaround is to temporarily remove this configuration from the Windows Registry in order to run the installer successfully.

In **HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\PowerShell**, ensure you have the following values set:

Key	Type	Value
EnableScripts	REG_DWORD	1
ExecutionPolicy	REG_SZ	Unrestricted

Modify the website configuration files if required

Advanced configuration of the websites is done by creating and editing configuration files in the installation folders. By default, on Windows, these files are in:

- SmartSpace website: **C:\Program Files (x86)\Ubisense 3\SmartSpace\Web\localsettings.json**
- REST API website: **C:\Program Files (x86)\Ubisense 3\WebApiCore\Web\localsettings.json**

These files should be created if you wish to make advanced configurations, rather than modifying the installed defaults in **appsettings.json**. This is because **appsettings.json** can be overwritten on a software upgrade.

Using OpenID Connect to enable external authentication

OpenID Connect support allows an external authentication authority to handle login and provide user ID and roles to SmartSpace Web. The external authority takes care of ensuring that the user is allowed to access the application, and sends claims about the user back to SmartSpace Web. SmartSpace can then be configured to extract the user ID, and optionally the roles, from the claims provided by the authority.

To use OpenID Connect, SmartSpace must be accessed via https, so make sure you have configured SSL in IIS.

There are two places where OpenID Connect must be configured: at the authority, and in SmartSpace Web.

Configuration at the Authority

The authority provides an application client ID and secret, and is configured to accept the redirect URIs that will be used during authentication. The authority will also be configured to control which users/groups are allowed to access SmartSpace, and which properties should be sent to SmartSpace for it to use as the user ID and/or roles. Generally this involves creating a new "application" instance or registration at the authority, and then configuring the application to

work with SmartSpace.

In order to ensure that only valid granted authorities are used, both the authority and SmartSpace share a configured "client id" and "client secret". The ID identifies the specific SmartSpace web site application at the authority, and the secret is used by SmartSpace to verify that the returned claims are valid. So the first step to setting up the authority is to create a new client ID in the authority, and to generate the client secret. The application must have the "code flow" or "code" response type enabled (also known as "Authorization code grant").

Authorities have different ways to do this, but for example, in Microsoft Azure Active Directory (AD), at the time of writing:

1. Go to App Registrations.
2. Click New registration.
3. Give the application a name and the set of users who can log in, and then click Register.
4. Click Certificates and secrets, and create a new secret.
5. Take a copy of the Value (it can only be viewed/copied on creation).

Set up the redirect URIs used during login and logout. These must match the redirect requests that the SmartSpace website generates when it attempts to login or logout a user by directing the browser to the authority.

- Sign in or callback URI: **https://<your server host as seen by the end user>/SmartSpace/signin-oidc**
- Sign out or front channel logout URI: **https://<your server host as seen by the end user>/SmartSpace/**

You can also define roles, or put a set of AD groups as claims. See the Azure AD documentation for how this is done.

Configuration in SmartSpace Web

There is an OpenID Connection configuration section in the web settings JSON files. Depending on your operating system this file is either **localsettings.json**, or **/etc/ubisense/web.json**.

```
"OpenIDC" : {
         "ClientId": "865b3f07-3f30-4881-895b-a831d11917ac",
         "ClientSecret": "<your secret shared with the authority goes here>",
         "MetadataAddress": "https://login.microsoftonline.com/fec9c4e9-d7de-4363-a4e3-
039b6f2606d2/v2.0/.well-known/openid-configuration",
         "IdClaim": "preferred_username",
         "RoleClaim": "http://schemas.microsoft.com/ws/2008/06/identity/claims/role",
         "LogoutURL": "",
         "DebugClaims": true
   },
```

The settings in this section are case sensitive, and have the following meanings:

Setting	Meaning
ClientId	The identity of this app at the OpenID Connect authority
ClientSecret	The shared secret used to generate and validate claims about the logged-in user
MetadataAddress	The OpenID Connect configuration meta-data document from your authority. For Azure AD, for example, go to Endpoints under your application registration, and copy "OpenID Connect metadata document".
IdClaim	The claim type to be used as the logged-in user in SmartSpace
RoleClaim	The claim type to be used as a role for the user in SmartSpace. Optional
LogoutURL	Some OpenID Connect authorities do not provide an "end_session_endpoint", so logout will be unsuccessful. In this case, provide a URL to log out the user from the client ID at the authority. Optional
DebugClaims	If set true, this will provide a page within SmartSpace Web that you can visit once authentication has redirected back to SmartSpace, and see the claims provided. The page can be found at SmartSpace/Auth/Claims. It is recommended you disable this configuration in production. The default is false. Optional

After the configuration has been saved, restart the web site. Go to IIS Manager -> Application Pools -> SmartSpaceCore -> Start.

You can test log in by visiting the SmartSpace Web site and clicking Login. You should be redirected to the authority login page. On successfully logging in, you will be asked to confirm sharing your profile with SmartSpace, and then be redirected back to SmartSpace Web. Note that if you are already logged in at the authority (for example if you are logged into Microsoft 365), there may be no need to provide any user/password – you may not even see the redirects as they can happen very quickly.

On success, the user ID should be displayed in the top left of the SmartSpace Web site. If this is not the case, enable DebugClaims, restart SmartSpace Web, and authenticate again. Then visit /SmartSpace/Auth/Claims to see what the authority returned, and pick a suitable IdClaim.

Using Roles from OpenID Connect

Claims returned by the authority that have a claim type configured in RoleClaim will be treated as user roles. These will be passed transparently through to SmartSpace. To use them within SmartSpace they should also be created as roles using the USERS / ROLES task in SmartSpace Config. You do not need to add members of the role within SmartSpace Config (though this is

allowed if necessary). Users that have the role according to the OpenID Connect authority will automatically be treated as having the role within SmartSpace Web.

For example, if the authority passes a claim "SmartSpace Supervisor", you should create a "SmartSpace Supervisor" role in the USERS / ROLES task in SmartSpace Config. After you have created the role within SmartSpace, you can:

- assign this role as a member of other roles
- specify the searches/views/properties that this role can access directly
- assign the role to reports, HMIs, and in ObjectView API views

Again, you can use *DebugClaims* to see what the authority has returned to SmartSpace for the currently authenticated user. Remember to disable DebugClaims and restart the web site before going into production.

Header Authentication (SiteMinder)

The websites can read the authenticated user from a header passed to them by a proxy server, such as SiteMinder. To configure this, set "AuthOptions/UserHeader" to be the name of the header from which the logged in user is to be extracted.

```
{
    "AuthOptions": {
       "UserHeader": "SITEMINDER_USER"
   }
}
```

If you configure this option, it is vital that users cannot access the website except through the proxy server, because otherwise they could add their own header as part of the request and gain unauthorized access. Typically in this case you would configure IIS bindings to only listen on the loopback interface, or configure IP Address and Domain Restrictions. See Microsoft IIS documentation for details.

Header Authentication with Other External Authentication Systems

Upstream reverse proxies can provide some other external authentication system, and pass both user and roles from that external system to SmartSpace.

AuthOptions.RolesHeader can be used with AuthOptions.UserHeader to specify a header from which to read roles for passthrough to SmartSpace. Roles are comma separated in the value of the

header provided. The config is optional: if not specified in the settings file then no roles will be passed through. The option is ignored if UserHeader is not set.

For example:

```
"AuthOptions": {
     "UserHeader": "AUTH_USER",
     "RolesHeader": "AUTH_ROLES",
      ...
 }
```

This would read the user (authenticated by the up-stream reverse proxy) from header AUTH_USER, and the roles from header AUTH_ROLES. If these are used, the web site should not be accessible except from the reverse proxy, which should filter out any user-supplied values for these headers.

As with roles in *OpenID Connect*, the roles that can be passed in the header should also be created in the USERS / ROLES task in SmartSpace Config so they can be assigned as members of other roles, or given searchs/views, etc.

Switching to Forms Authentication

By default the Windows website installation uses Windows authentication for login. You can switch to forms authentication by configuring LDAP parameters and setting up "AuthOptions/UseCookiesOnWindows". For production or integration deployment, you will need an SSL certificate signed by a suitable root authority for your users. For development or test deployment, a test certificate can be used instead (e.g. generated locally using OpenSSL). If you configure forms authenticatication without SSL, it will not work.

In the following examples, the first example shows the configuration for an Active Directory server, whilst the second shows the configuration for an LDAP server that does not require a login for searching.

LDAP authentication with an Active Directory server

```
"LDAPAuth": {
      "Server": "adserver.company.com",
      "Port": "389",
      "User": "",
      "Password": "",
      "SearchStart": "dc=company,dc=com",
      "AccountId": "sAMAccountName"
   },

   "AuthOptions": {
      "UseCookiesOnWindows" : true,
      "ExpiryTimeSpan": "00:30",
      "SlidingExpiry":  true
   }
}
```

LDAP authentication with no login for searching

```
"LDAPAuth": {
      "Server": "ldap.company.com",
      "Port": "389",
      "User": "",
      "Password": "",
      "SearchStart": "ou=people,dc=company,dc=com",
      "AccountId": "uid",
      "ObjectClass": "account"
   },

   "AuthOptions": {
      "UseCookiesOnWindows" : true,
      "ExpiryTimeSpan": "00:30",
      "SlidingExpiry":  true
   }
}
```

Note: The example given above works with the default OpenLDAP schema: other servers and schema might require different parameters.

How the LDAP validator behaves

The LDAP validator does the following:

1. If User option set, bind using this user/password.
2. If SearchStart is set, use the SearchStart, AccountId and ObjectClass to search for the DN of the entered user name.
3. Bind using the DN, if the search succeeded, or the entered user name. Use the entered password. If this bind succeeds, the user is authenticated successfully.

The validator reports what it is doing on the website trace (always on). for example:

```
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: Is user valid
marcin
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: searching (&
(objectclass=nsAccount)(uid=marcin)) at base ou=people,dc=ubisense,dc=aws
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: binding as user
uid=marcin,ou=People,dc=ubisense,dc=aws
```

Enabling Secure LDAP (LDAPS)

To enable secure LDAP (LDAPS) for the connection you must use port 636. By using this port, SSL/TLS is enabled for the connection to the LDAP server (all other port numbers use TCP).

```
"LDAPAuth": {
      "Server": "adserver.company.com",
      "Port": "636",
      "User": "",
      "Password": "",
      "SearchStart": "dc=company,dc=com",
      "AccountId": "sAMAccountName"
   },

   "AuthOptions": {
      "UseCookiesOnWindows" : true,
      "ExpiryTimeSpan": "00:30",
      "SlidingExpiry":  true
   }
}
```

Configuring the Authentication Timespan

The cookies authentication uses an expiry time of 30 minutes and a sliding timespan. This means that the authentication will expire 30 minutes after the user closes the website, but will continue to be refreshed while the user is still visiting the website.

You can disable this sliding expiry and set an absolute time after which login will need to be repeated. For example, to log out after three hours:

```
{
   "AuthOptions": {
      "ExpiryTimeSpan": "03:00",
      "SlidingExpiry":  false
   }
}
```

Disable Hardened Headers

By default the website injects headers in each response for penetration security. These disable cross-site/cross-frame scripting, prevent content type sniffing, etc. If necessary, these headers

can be disabled, and IIS configured manually to add appropriate headers instead.

```
{
    "SecurityOptions": {
       "HardenHeaders": false
    }
}
```

Adding a custom theme for the website

You can customize the look of the SmartSpace website to better reflect your corporate identity, for example by replacing the Ubisense logo with your own or using a custom stylesheet. To prevent such customization from being overwritten during website upgrades, you should store your local theme in an external folder on the host machine, for example **C:\Ubisense\localtheme**. You specify the folder using the website configuration setting ContentOptions / LocalThemePath.

```
{
    "ContentOptions": {
       "LocalThemePath": "C:\\Ubisense\\localtheme\\"
    }
}
```

If the custom theme folder contains any **.min.css** files, they are loaded in place of the **wwwroot/bundle/base.css**. If the folder contains **custom.css**, it will be loaded in addition to other **css** files.

The following files can also be overridden by adding corresponding files in the custom theme folder:

- **images/logo.png**
- **images/background.png**
- **images/ubisense_large.png**
- **images/ubisense_small.png**
- **manifest.json**

REST API Security

Enable Cross Origin Scripting

CORS is supported for the SmartSpace REST API. For features using SmartSpace Web where CORS is not supported, there are workarounds such as making configuration changes so that the

browser treats localhost requests and the remote site as if they come from the same domain, disabling hardened headers (see above), or using IIS or Apache.

By default, no CORS headers are sent, so browsers will refuse to execute the API web methods from a page served from a different web server.

The option AllowOrigins in the **appsettings.json** file which overrides settings in **localsettings.json** enables cross origin scripting:

```
{
...
   "SecurityOptions": {
   "HardenHeaders": true,
   "AllowOrigins": [ "http://example.com", "https://*.mydomain.com" ]
   }
...
}
```

If AllowOrigins is set, and matches the origin of a request, the API will respond with suitable headers, for example:

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: X-Requested-With
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Origin: https://server.mydomain.com
Access-Control-Max-Age: 3600
```

The browser will now allow the request. Note that this allows the browser to cache the pre-flight OPTIONS responses for up to an hour, to reduce load on the API server. Thus changes to the allowed origins may not be picked up by browsers for an hour.

Bearer Authentication

From version 3.8 SP5, the SmartSpace Application REST API supports Bearer authentication. The two OAuth flows that Ubisense support are Authorization Code Flow with PKCE and Client Credentials flow.

Depending on the OAuth method that you use, you need to set up either a role or scope when configuring your authentication. If you are using the PKCE flow, you need to add a role called "smartspace-rest-api-access-role". Only users that are members of this role can access the REST API. If you are using the Client Credentials flow, you need to add a scope called "smartspace-api-scope". Only applications that are added to this scope can access the REST API.

Configuring the Application REST API

Once you have configured your authentication, you will have the basic information needed to configure the settings in the REST API. On Linux, these are configured in /etc/ubisense/restapi.json. On Windows, they are in localsettings.json in the REST API installation folder; this is normally C:\Program Files (x86)\Ubisense 3\WebApiCore\Web.

The following sections are needed in the REST API settings:

- AllowHosts, to support CORS headers and allow access from external web pages
- JWTAuth, to configure the application information, and
- AuthOptions, to require authentication for all access

An example of the REST API settings is shown below. Replace the following parameters with values from your authentication configuration:

- metadataAddress: The URL where your OpenID Connect configuration is published.
- authority: The base URL of your authorization server.
- audience: The identifier for your API that clients must include in their token requests.
- nameClaim: The claim in the token that contains the username.
- validIssuers: An array of issuer URLs that are considered valid. Tokens from other issuers will be rejected.
- validateLifetime: A boolean value indicating whether the token's lifetime should be validated.
- requireHttpsMetadata: A boolean value indicating whether HTTPS is required for the metadata endpoint.

```
"JWTAuth": {
  "metadataAddress": "https://<your-auth-server>/path/to/.well-known/openid-configuration",
  "authority": "https://<your-auth-server>",
  "audience": "<your-api-identifier>",
  "nameClaim": "preferred_username",
  "validIssuers": ["https://<your-auth-server>/path/to/issuer"],
  "validateLifetime": true,
  "requireHttpsMetadata": false
}
```

The parameter `"validIssuers":` should be the list of valid issuers. For multi-tenant organizations, you may configure multiple valid issuers with different tenant IDs.

Once you have saved the settings, restart the REST API service, if using Linux, or the SmartSpaceCoreApi application pool, if using Windows/IIS.

Fetching the token

After configuring the REST API settings, you can fetch and use the JWT. The following code snippet shows an example of fetching and using the token for the Client Credentials flow in C#.

Fetching the token

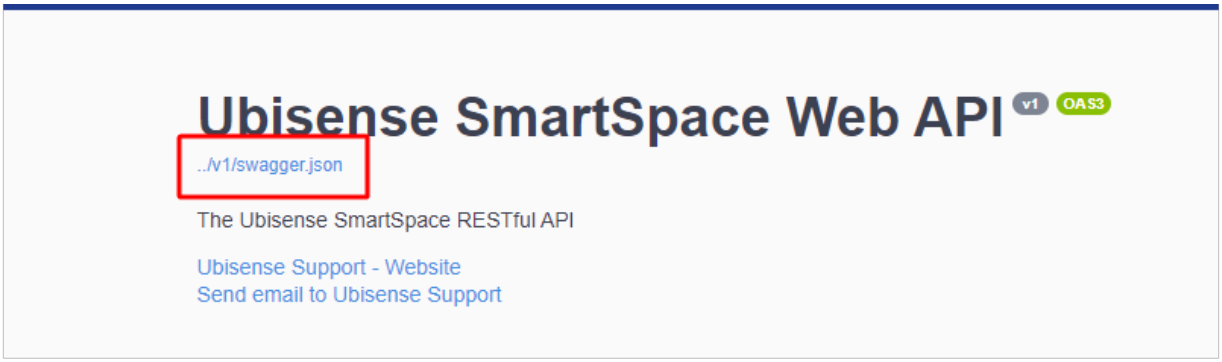
```
internal class AuthenticationService{
    private readonly HttpClient _httpClient;
    public AuthenticationService(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }
    public async Task<string> GetAccessTokenAsync(string clientId, string clientSecret, Uri tokenEndpointUri)
    {
        var clientCredentials = new FormUrlEncodedContent(new[]
        {
        new KeyValuePair<string, string>("grant_type", "client_credentials"),
        new KeyValuePair<string, string>("client_id", clientId),
        new KeyValuePair<string, string>("client_secret", clientSecret),
    });
        var response = await _httpClient.PostAsync(tokenEndpointUri, clientCredentials);
        response.EnsureSuccessStatusCode();
        var content = await response.Content.ReadAsStringAsync();
        var tokenResponse = JsonSerializer.Deserialize<Dictionary<string, object>>(content);
        return tokenResponse["access_token"].ToString();
    }
  }
```

Using the returned token

```
var accessToken = await _authenticationService.GetAccessTokenAsync(
    "sample-console-app",
    "<client_secret>",//Replace this with the client secret. This is a secret and should be treated as such in production scenarios
    new Uri("<token-endpoint-url>") // Replace with your token endpoint url
  );
  _httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", accessToken);
  const string apiUrl = "http://localhost:5001/api/TagTypes";
```

```
    var response = await _httpClient.GetAsync(apiUrl);
    if (response.IsSuccessStatusCode)
    {
        var content = await response.Content.ReadAsStringAsync();
        Console.WriteLine("Response Content:\n" + content);
    }
    else
    {
        Console.WriteLine($"Error: {response.StatusCode}");
    }
```

An API specification JSON file is provided which can be used to generate your client code. The file can be found at http://localhost/smartspaceapi and clicking the link under the title.



Errors

When the website is loaded, you may see 502.5 “ANCM Out-of-Process Startup Failure”.

This is normally because the .NET Runtime could not be found. Make sure you restarted the server after installing the ASP.NET Core Runtime 6.0.x Hosting bundle.

Also, make sure that you included the Security/Windows Authentication feature when deploying IIS, as this is required by the websites on Windows unless forms or header authentication is configured.

Security Configuration for SmartSpace Web with Security Manager

If you are using a non-trivial security manager configuration to force authentication for services (as is the case for ACS installations) then you must run the **ubisense_cache_service_credentials**

tool on the web server host. This is because the **credentials.dat** file created by the tool (in the latest version) allows IIS_IUSRS as a reader. Without this, the web site code cannot read the credentials, and therefore cannot connect to the platform services it needs.

For a Windows server, you *must* use the version of the **ubisense_cache_service_credentials** tool from the 3.4 sp1 distribution or above.

Installing SmartSpace Web on Linux

Because of the variations between Linux distributions and between package management systems used by different Enterprise configurations, Ubisense do not provide an automatic installation method for Linux. Instead the instructions describe the necessary state of a Ubisense installation on Linux, prerequisites for the installation, the layout and permissions expected, and example scripts. In this section, we will describe configuring the websites using Apache2 as a reverse proxy. *Advanced configuration* options will then be covered later.

Linux Requirements for SmartSpace Web

We support recent enterprise Linux distributions, such as SUSE Linux Enterprise Server11+ or Red Hat® Enterprise Linux®v7+.

The following instructions assume you are configuring a reverse proxy in Apache 2.4.23 or above. For Red Hat® Enterprise Linux® Apache 2.4.23 is only available for version 8+. Whilst configuring a reverse proxy on earlier versions of Red Hat® Enterprise Linux® is possible, instructions for this are beyond the scope of this guide.

The following packages must be installed on the server.

- ldap 2.4 libraries

For production or integration deployment, you will need an SSL certificate signed by a suitable root authority for your users. This certificate will be installed for Apache2. SSL (TLS) is required for Linux installation because the website uses forms-based authentication, and so must have transport level security configured. For development or test deployment, a test certificate can be used instead (e.g. generated locally using OpenSSL).

Microsoft .NET Runtimes

We *strongly recommend* you install the latest ASP.NET Core Runtime 6.0.x. For links to the requisite downloads, see *https://dotnet.microsoft.com/en-us/download/dotnet/6.0*.

If Microsoft .NET is already installed on the server, the websites will use that runtime provided it is the version of .NET required by that release of SmartSpace. If a server-wide installation of .NET is *not* provided, then a Ubisense-supplied isolated local runtime will be used, shared only by the platform services that use it. It is important to note that if a version of .NET Core is installed that is different to the version required by SmartSpace (for example 3.1 if SmartSpace wanted 6.0), then the website will not work, and there is no fallback to the isolated local runtime.

If Microsoft .NET is installed on a Linux server, and is *not* in the default path of **/usr/share/dotnet**, then it is necessary to ensure that the environment variable **DOTNET_ROOT** is set to point to where the .NET runtime is located in your Linux file system. This is usually added to the user profiles automatically, but may not be available to systemd services, so if running core and controller from systemd, it is important to add **DOTNET_ROOT** to the systemd service definitions. See the sample systemd scripts in *Installing the Server Software on Linux* in your installation guide for an example.

Server Configuration

Ensure the web server is connected to the platform, using multicast, unicast cluster (see Smart Space Using Unicast Cluster Setup Guide on the Ubisense Documentation Portal), or a site connector.

If you have not already done so, on the web server install the platform servers for Linux, configure a dataset directory, and start a local controller. The website will be deployed on this controller. See *Installing the Server Software on Linux* in your installation guide.

Authentication Options

There are two authentication methods supported in the web sites when deployed on Linux: Forms authentication, and Header authentication.

Forms authentication directs the user to a login page when they attempt to access a part of the web sites that requires authentication. This login page gathers the user and password, which are then validated using a configured LDAP server. If this succeeds, a cookie is returned to the user's browser, which is used to authenticate in subsequent requests. Forms authentication requires the web site to be accessed via HTTPS for all authenticated or login traffic, to protect the credentials and cookies. This is configured in the reverse proxy that handles the HTTPS protocol.

Header authentication relies on another system, such as SiteMinder, doing authentication before passing the request to the website. A special header is set by this up-stream system on each request that reaches the web site, indicating the authenticated user. The web site simply assumes that the given header is authoritative. This is a commonly used method in enterprise environments.

Either method can be used for the SmartSpace website, but the Rest API only supports Header authentication (or no authentication).

Configuration Files

Set up the configuration files for the SmartSpace website and REST API. On Linux these are placed in **/etc/ubisense**. The files should all be readable only by the user that runs the platform controller. The following configuration files are used:

web.json

This contains configuration specific to the website. In the following examples, we will set up the parameters to access the LDAP server used to validate the user's credentials. We also set a proxy base path matching the reverse proxy path we will configure in Apache2 for the website. The first example shows the configuration for an Active Directory server, whilst the second shows the configuration for an LDAP server that does not require a login for searching.

LDAP authentication with an Active Directory server

```
{
    "LDAPAuth": {
      "Server": "adserver.company.com",
      "Port": "389",
      "User": "",
      "Password": "",
      "SearchStart": "dc=company,dc=com",
      "AccountId": "sAMAccountName"
   },

     "ProxyOptions": {
        "Base": "/SmartSpace"
    }
 }
```

LDAP authentication with no login for searching

```
{
    "LDAPAuth": {
      "Server": "ldap.company.com",
      "Port": "389",
      "User": "",
      "Password": "",
      "SearchStart": "ou=people,dc=company,dc=com",
      "AccountId": "uid",
      "ObjectClass": "account"
   },


     "ProxyOptions": {
        "Base": "/SmartSpace"
    }
 }
```

Note: The example given above works with the default OpenLDAP schema: other servers and schema might require different parameters.

How the LDAP validator behaves

The LDAP validator does the following:

1. If User option set, bind using this user/password.
2. If SearchStart is set, use the SearchStart, AccountId and ObjectClass to search for the DN of the entered user name.
3. Bind using the DN, if the search succeeded, or the entered user name. Use the entered password. If this bind succeeds, the user is authenticated successfully.

The validator reports what it is doing on the website trace (always on). for example:

```
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: Is user valid
marcin
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: searching (&
(objectclass=nsAccount)(uid=marcin)) at base ou=people,dc=ubisense,dc=aws
[Tue Sep 24 14:23:03 2019, 127.0.0.1:42943] website: LDAPValidator: binding as user
uid=marcin,ou=People,dc=ubisense,dc=aws
```

Enabling Secure LDAP (LDAPS)

To enable secure LDAP (LDAPS) for the connection you must use port 636. By using this port, SSL/TLS is enabled for the connection to the LDAP server (all other port numbers use TCP).

Note: If the LDAP server uses LDAPS (port 636), then the web site server must be able to verify the certificate that the LDAP server uses. For example, in active directory, the root certificate used to sign the LDAP server certificate should be imported into the web server host as a certificate authority (CA). How this is done depends on the specific operating system, but if it is not done, then LDAPS authentication will fail.

```
"LDAPAuth": {
      "Server": "adserver.company.com",
      "Port": "636",
      "User": "",
      "Password": "",
      "SearchStart": "dc=company,dc=com",
      "AccountId": "sAMAccountName"
   },

   "AuthOptions": {
      "UseCookiesOnWindows" : true,
      "ExpiryTimeSpan": "00:30",
      "SlidingExpiry":  true
   }
}
```

Using OpenID Connect to enable external authentication

OpenID Connect support allows an external authentication authority to handle login and provide user ID and roles to SmartSpace Web. The external authority takes care of ensuring that the user is allowed to access the application, and sends claims about the user back to SmartSpace Web. SmartSpace can then be configured to extract the user ID, and optionally the roles, from the claims provided by the authority.

To use OpenID Connect, SmartSpace must be accessed via https, so make sure you have configured SSL in your reverse proxy.

There are two places where OpenID Connect must be configured: at the authority, and in SmartSpace Web.

Configuration at the Authority

The authority provides an application client ID and secret, and is configured to accept the redirect URIs that will be used during authentication. The authority will also be configured to control which users/groups are allowed to access SmartSpace, and which properties should be sent to SmartSpace for it to use as the user ID and/or roles. Generally this involves creating a new "application" instance or registration at the authority, and then configuring the application to work with SmartSpace.

In order to ensure that only valid granted authorities are used, both the authority and SmartSpace share a configured "client id" and "client secret". The ID identifies the specific SmartSpace web site application at the authority, and the secret is used by SmartSpace to verify that the returned claims are valid. So the first step to setting up the authority is to create a new client ID in the authority, and to generate the client secret. The application must have the "code flow" or "code" response type enabled (also known as "Authorization code grant").

Authorities have different ways to do this, but for example, in Microsoft Azure Active Directory (AD), at the time of writing:

1. Go to App Registrations.
2. Click New registration.
3. Give the application a name and the set of users who can log in, and then click Register.
4. Click Certificates and secrets, and create a new secret.
5. Take a copy of the Value (it can only be viewed/copied on creation).

Set up the redirect URIs used during login and logout. These must match the redirect requests that the SmartSpace website generates when it attempts to login or logout a user by directing the browser to the authority.

- Sign in or callback URI: **https://<your server host as seen by the end user>/SmartSpace/signin-oidc**
- Sign out or front channel logout URI: **https://<your server host as seen by the end user>/SmartSpace/**

You can also define roles, or put a set of AD groups as claims. See the Azure AD documentation for how this is done.

Configuration in SmartSpace Web

There is an OpenID Connection configuration section in the web settings JSON files. Depending on your operating system this file is either **localsettings.json**, or **/etc/ubisense/web.json**.

```
"OpenIDC" : {
         "ClientId": "865b3f07-3f30-4881-895b-a831d11917ac",
         "ClientSecret": "<your secret shared with the authority goes here>",
         "MetadataAddress": "https://login.microsoftonline.com/fec9c4e9-d7de-4363-a4e3-
039b6f2606d2/v2.0/.well-known/openid-configuration",
         "IdClaim": "preferred_username",
         "RoleClaim": "http://schemas.microsoft.com/ws/2008/06/identity/claims/role",
         "LogoutURL": "",
         "DebugClaims": true
   },
```

The settings in this section are case sensitive, and have the following meanings:

Setting	Meaning
ClientId	The identity of this app at the OpenID Connect authority
ClientSecret	The shared secret used to generate and validate claims about the logged-in user
MetadataAddress	The OpenID Connect configuration meta-data document from your authority. For Azure AD, for example, go to Endpoints under your application registration, and copy "OpenID Connect metadata document".
IdClaim	The claim type to be used as the logged-in user in SmartSpace
RoleClaim	The claim type to be used as a role for the user in SmartSpace. Optional
LogoutURL	Some OpenID Connect authorities do not provide an "end_session_endpoint", so logout will be unsuccessful. In this case, provide a URL to log out the user from the client ID at the authority. Optional
DebugClaims	If set true, this will provide a page within SmartSpace Web that you can visit once authentication has redirected back to SmartSpace, and see the claims provided. The page can be found at SmartSpace/Auth/Claims. It is recommended to disable this configuration in production. The default is false. Optional

After the configuration has been saved, restart the web site by using Service Manager to restart the Ubisense/Visibility/Web site service.

You can test log in by visiting the SmartSpace Web site and clicking Login. You should be redirected to the authority login page. On successfully logging in, you will be asked to confirm sharing your profile with SmartSpace, and then be redirected back to SmartSpace Web. Note that if you are already logged in at the authority (for example if you are logged into Microsoft 365), there may be no need to provide any user/password – you may not even see the redirects as they can happen very quickly.

On success, the user ID should be displayed in the top left of the SmartSpace Web site. If this is not the case, enable DebugClaims, restart SmartSpace Web, and authenticate again. Then visit /SmartSpace/Auth/Claims to see what the authority returned, and pick a suitable IdClaim.

Using Roles from OpenID Connect

Claims returned by the authority that have a claim type configured in RoleClaim will be treated as user roles. These will be passed transparently through to SmartSpace. To use them within SmartSpace they should also be created as roles using the USERS / ROLES task in SmartSpace Config. You do not need to add members of the role within SmartSpace Config (though this is allowed if necessary). Users that have the role according to the OpenID Connect authority will

automatically be treated as having the role within SmartSpace Web.

For example, if the authority passes a claim "SmartSpace Supervisor", you should create a "SmartSpace Supervisor" role in the USERS / ROLES task in SmartSpace Config. After you have created the role within SmartSpace, you can:

- assign this role as a member of other roles
- specify the searches/views/properties that this role can access directly
- assign the role to reports, HMIs, and in ObjectView API views

Again, you can use *DebugClaims* to see what the authority has returned to SmartSpace for the currently authenticated user. Remember to disable DebugClaims and restart the web site before going into production.

Header Authentication with Other External Authentication Systems

Upstream reverse proxies can provide some other external authentication system, and pass both user and roles from that external system to SmartSpace.

AuthOptions.RolesHeader can be used with AuthOptions.UserHeader to specify a header from which to read roles for passthrough to SmartSpace. Roles are comma separated in the value of the header provided. The config is optional: if not specified in the settings file then no roles will be passed through. The option is ignored if UserHeader is not set.

For example:

```
"AuthOptions": {
     "UserHeader": "AUTH_USER",
     "RolesHeader": "AUTH_ROLES",
      ...
 }
```

This would read the user (authenticated by the up-stream reverse proxy) from header AUTH_USER, and the roles from header AUTH_ROLES. If these are used, the web site should not be accessible except from the reverse proxy, which should filter out any user-supplied values for these headers.

As with roles in *OpenID Connect*, the roles that can be passed in the header should also be created in the USERS / ROLES task in SmartSpace Config so they can be assigned as members of other roles, or given searchs/views, etc.

restapi.json

This contains configuration specific to the REST API. In this example, we will set up a proxy base path matching the reverse proxy path we will configure in Apache2 for the API:

```
{
    "ProxyOptions": {
        "Base": "/SmartSpaceApi"
   }
}
```

Here we are allowing anonymous access to the API. For header authentication see the *advanced configuration*.

shared.json

shared.json is a configuration file that is loaded by both web sites, where options shared by the two can be set up. This is not used in our example configuration.

Deploy the Platform Services

The platform services should be deployed using the Service Manager client or the **ubisense_installer** command-line tool.

To deploy the services using Service Manager:

1. Run Service Manager 3 on a Windows client connected to the platform.
2. Click **Install services...**, click **Browse** and navigate to the extracted SmartSpace release, then go to the **web\linux\packages** folder.
3. If you already have a .NET Core Runtime installed on the web server, unselect **Shared runtime** in the list of features to install.
4. Click **Install** and the services are installed. You can click **Finish** when the "Service installation complete" message displays to return to the main Service Manager screen.
5. If you have multiple Linux controllers, deploy the website services onto the prepared Linux server:
    a. Expand the **Controllers** entry under HIERARCHY, so you can see the web server controller.
    b. Expand the **Services** entry under HIERARCHY, find the "Ubisense/Visibility/Web site" service. Drag this service and drop it onto the web server controller. It should deploy

onto that controller.

c. Repeat the above for the service "Ubisense/Application integration/RestAPI site".

6. The Web site and Rest API site services should now be running on the Linux web server, but will not be visible from the wider network.

To deploy the services using the **ubisense_installer** command-line tool:

1. Go to the **web\linux\packages** directory of your SmartSpace distribution directory.
2. Run the following commands to install and deploy the Web site and REST API site:

```
ubisense_installer -ud SmartSpaceWeb.xml
```

```
ubisense_installer -ud SmartSpaceRestApi.xml
```

3. If you do not have a .NET Core Runtime installed on the web server, you must also run the following command:

```
ubisense_installer -ud SharedRuntime.xml
```

Configure Apache2 Reverse Proxy

Now install and configure Apache2 as a reverse proxy. The instructions below are targeted at SLES, and will need to be adapted for other Linux distributions.

Install Apache2

Use the package management software for your Linux distribution to install Apache2. For example:

```
sudo zypper in apache2
```

You will also need to ensure all required Apache modules are enabled using the following command:

```
sudo a2enmod <module name>
```

The required modules include:

```
mod_proxy
mod_proxy_http
mod_ssl
mod_headers
mod_rewrite
```

On some variants of Linux, drop the "mod_" prefix when enabling a module for Apache. You can see the list of enabled modules with the command:

```
sudo apache2ctl -M
```

Modules may be displayed with slightly different names in the output generated by this command, with a **_module** suffix instead of a **mod_** prefix.

Enable Outbound Connections

For SELinux servers such as RHEL 7, you may find that the Apache service cannot connect to another website. To correct this you need to enable outbound connections by running the following command:

```
/usr/sbin/setsebool -P httpd_can_network_connect 1
```

Install the SSL certificates

Place the SSL certificate and key in a suitable location:

```
/etc/apache2/ssl.crt/localhost.crt
/etc/apache2/ssl.key/localhost.key
```

Creating the service configuration file

Apache is configured by **.conf** files located in **/etc/apache2/vhosts.d/**.

Create a file **smartspace.conf**. The example below matches the proxy paths configured in the **web.json** and **restapi.json** example files above.

```
<VirtualHost *:*>
        RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}
</VirtualHost>

<VirtualHost *:80>
    # Rewrite http to https
    RewriteEngine On
    RewriteCond %{HTTPS} !=on
    RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
</VirtualHost>

<VirtualHost *:443>
        SSLProxyEngine on
        ProxyPreserveHost On

        # We proxy SmartSpaceApi to the REST api http port
        ProxyPass /SmartSpaceApi http://127.0.0.1:5002/SmartSpaceApi
        ProxyPassReverse /SmartSpaceApi http://127.0.0.1:5002/SmartSpaceApi

        # We proxy SmartSpace to the web site http port
        ProxyPass /SmartSpace http://127.0.0.1:5000/SmartSpace
        ProxyPassReverse /SmartSpace http://127.0.0.1:5000/SmartSpace

        # Add the forwarded protocol header
        RequestHeader set "X-Forwarded-Proto" expr=%{REQUEST_SCHEME}

       # Using localhost as server hostname
        ServerName mywebserverhost.domain.com
        ServerAlias mywebserverhost

        # Set logging_dir to a suitable logging directory
        ErrorLog /var/log/apache2/smartspace_error.log
        CustomLog /var/log/apache2/smartspace_custom.log common

        # Give away as little as possible on 404.
        ErrorDocument 404 "Not found"

        # Redirect top level to the SmartSpace web site
        RedirectMatch ^/$ /SmartSpace

        # Configure the https options to be suitably secure
        SSLEngine on
        SSLProtocol all -SSLv2
        SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:!RC4+RSA:+HIGH:+MEDIUM:!LOW:!RC4
        SSLCertificateFile /etc/apache2/ssl.crt/localhost.crt
        SSLCertificateKeyFile /etc/apache2/ssl.key/localhost.key
</VirtualHost>
```

In this example, **SSLCertificateFile** should be the primary certificate file for the domain name. **SSLCertificateKeyFile** should be the key file generated when CSR is created. **SSLCertificateChainFile** should be the intermediate certificate file (if any) that was supplied by the certificate authority.

Enable SSL for Apache

Additionally SSL must be enabled for Apache by adding the **SSL** flag to **APACHE_SERVER_FLAGS** in **/etc/sysconfig/apache2**:

```
APACHE_SERVER_FLAGS="SSL"
```

You will need to restart Apache to reload any new or changed configuration files, using the following commands:

```
sudo systemctl restart apache2
sudo systemctl enable apache2
```

Check the status of Apache with the command:

```
sudo systemctl status apache2
```

Test the website

Visit the website in a browser. You should be redirected to the top level SmartSpace website page.

Advanced Configuration

Header Authentication (SiteMinder)

The websites can read the authenticated user from a header passed to them by a proxy server, such as SiteMinder. To configure this, in **shared.json**, **web.json** or **restapi.json**, set **AuthOptions/UserHeader** to be the name of the header from which the logged in user is to be extracted. If this option is configured, then the LDAPAuth options do not need to be set.

```
{
    "AuthOptions": {
       "UserHeader": "SITEMINDER_USER",
       "RequireAuth": true
   }
}
```

If UserHeader has been configured, it would be normal to also require authentication for all pages. This is what the RequireAuth configuration setting does. Since the UserHeader assumes

that the header passed in the request has been checked, it is important that the user should not be able to bypass the proxy server and pass their own user header directly to the web sites.

Configuring the Authentication Timespan

The default authentication on Linux uses an expiry time of 30 minutes and a sliding timespan. This means that the authentication will expire 30 minutes after the user closes the website, but will continue to be refreshed while the user is still visiting the website.

You can disable this sliding expiry and set an absolute time after which login will need to be repeated. For example, to log out after three hours:

```
{
   "AuthOptions": {
      "ExpiryTimeSpan": "03:00",
      "SlidingExpiry":  false
   }
}
```

Disable Hardened Headers

By default the website injects headers in each response for penetration security. These disable cross-site/cross-frame scripting, prevent content type sniffing, etc. If necessary, these headers can be disabled, and the reverse proxy configured manually to add appropriate headers instead.

```
{
    "SecurityOptions": {
       "HardenHeaders": false
    }
}
```

Enable Cross Origin Scripting

CORS is supported for the SmartSpace REST API. For features using SmartSpace Web where CORS is not supported, there are workarounds such as making configuration changes so that the browser treats localhost requests and the remote site as if they come from the same domain, disabling hardened headers (see above), or using IIS or Apache.

By default, no CORS headers are sent, so browsers will refuse to execute the API web methods from a page served from a different web server.

The option AllowOrigins in the **appsettings.json** file which overrides settings in **localsettings.json** enables cross origin scripting:

```
{
...
   "SecurityOptions": {
   "HardenHeaders": true,
   "AllowOrigins": [ "http://example.com", "https://*.mydomain.com" ]
   }
...
}
```

If AllowOrigins is set, and matches the origin of a request, the API will respond with suitable headers, for example:

```
Access-Control-Allow-Credentials: true
Access-Control-Allow-Headers: X-Requested-With
Access-Control-Allow-Methods: PUT
Access-Control-Allow-Origin: https://server.mydomain.com
Access-Control-Max-Age: 3600
```

The browser will now allow the request. Note that this allows the browser to cache the pre-flight OPTIONS responses for up to an hour, to reduce load on the API server. Thus changes to the allowed origins may not be picked up by browsers for an hour.

Adding a custom theme for the website

You can customize the look of the SmartSpace website to better reflect your corporate identity, for example by replacing the Ubisense logo with your own or using a custom stylesheet. To prevent such customization from being overwritten during website upgrades, you should store your local theme in an external folder on the host machine, for example **/home/platform/localtheme**. You specify the folder using the website configuration setting ContentOptions / LocalThemePath.

```
{
    "ContentOptions": {
       "LocalThemePath": "//home//platform//localtheme//"
    }
}
```

If the custom theme folder contains any **.min.css** files, they are loaded in place of the **wwwroot/bundle/base.css**. If the folder contains **custom.css**, it will be loaded in addition to other **css** files.

The following files can also be overridden by adding corresponding files in the custom theme folder:

- **images/logo.png**
- **images/background.png**
- **images/ubisense_large.png**
- **images/ubisense_small.png**
- **manifest.json**

Removing the Shared Runtime after Undeploying the Websites

If you deploy the website and/or the REST API without installing the .NET Runtime on the server, and subsequently move these website services to another controller, the shared runtime will be left in the dataset. You can simply delete this directory when it is no longer needed by any locally deployed services. The folder is:

```
<dataset path>/Ubisense/Platform/Shared\ runtime/x.x.x/
```

where `x.x.x` is the .NET Runtime version number.

Security Configuration for SmartSpace Web with Security Manager

If you are using a non-trivial security manager configuration to force authentication for services (as is the case for ACS installations) then you must run the **ubisense_cache_service_credentials** tool on the web server host. This is because the **credentials.dat** file created by the tool (in the latest version) allows IIS_IUSRS as a reader. Without this, the web site code cannot read the credentials, and therefore cannot connect to the platform services it needs.