

# **Authorization Server**

## **Version 3.1.3**

---

### **User Guide**

*20.02.2024*

*TCG Informatik AG*

## Table of contents

---

1. Getting Started	3
1.1 System Requirements	3
1.2 Understanding the User Interface	6
1.3 How to get Started	9
1.4 Manually registering clients (applications) to the Authorization Service	11
2. Concepts	14
2.1 Platform Authorization and Authentication	14
3. Advanced Topics	17
3.1 External Identity Providers	17
3.2 Using Entra ID as an External Identity Provider	23
3.3 Writing Plugins for the Authorization Server	26
3.4 Certificates	28
3.5 Application Settings explained in detail	30
3.6 Using Postman to Test Authentication Flows	34
4. About	37
4.1 Release Notes	37
4.2 Copyright	40

# 1. Getting Started

---

## 1.1 System Requirements

---

These are the system requirements for TCG Authorization Server.

### 1.1.1 Operating System

---

The following operating systems and system components are required.

#### Supported

Microsoft Windows Server 2016 or higher

#### Recommended

Microsoft Windows Server 2019

### 1.1.2 Microsoft Windows Server Roles and Features

---

To install and set up the TCG Authorization Server, you must first install Internet Information Services (IIS) and the .NET 6 hosting bundle for IIS.

**Important** Please make sure that the latest updates for Microsoft Windows and all the installed components are installed.

#### Setting Up IIS

If the IIS features are not installed yet on your system, you can use the following commands in a PowerShell to install them:

```
Set-ExecutionPolicy Bypass -Scope Process
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerRole
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServer
Enable-WindowsOptionalFeature -Online -FeatureName IIS-WebServerManagementTools
Enable-WindowsOptionalFeature -Online -FeatureName IIS-ManagementConsole
```

#### .NET 6 Hosting Bundle for IIS

**Important** The installation of the .NET 6 SDK or .NET 6 Runtime are **not enough**, <https://dotnet.microsoft.com/en-us/download/dotnet/6.0>

#### Configured SSL Certificates

A configured HTTPS certificate is mandatory. Browsers refuse to use required encryption modules in case the communication is not encrypted.

## Extended IIS Maximum Query String Length

By default, IIS limits the maximum query string length on IIS to 2048 bytes. This can lead to requests to the Authorization Server being blocked by IIS. Therefore, Authorization Server ships with an extended query length limitation to 10000 bytes.

If this limit is still not sufficient, please notify the vendor. Increase the **maximum query string (Bytes)** in the Request Filtering Settings of IIS. Alternatively, you can set it in the IIS web.config file of the Authorization Server, however, this can lead to installations not being able to update that file anymore.

```
<security>
  <requestFiltering>
    <requestLimits maxQueryString="100000" />
  </requestFiltering>
</security>
```

## 1.1.3 Encrypted Communication (HTTPS)

It is not possible to run the TCG Authorization Server without SSL encrypted web traffic. The transferred data is very confidential and leaking this sort of data exposes access to the entire set of services that depend on it. The only case where non-encrypted http traffic is considered ok is in a load balancer scenario where the IT specialist has the necessary experience to create private networks. All known requirements for SSL certificates also apply, such as they must not be expired, state-of-the-art ciphers, self-signed certificates require explicit setup to trusted root certificates.

## 1.1.4 Hardware Requirements

The following hardware requirements need to be considered.

### Storage

A solid-state drive (SSD); no spinning disks

### Memory and CPU

- TCG Authorization Server - at minimum 4 GB RAM, 2 cores
- Database server - at minimum 4 GB RAM, 2 cores, optimized for I/O operations

## 1.1.5 Supported Database Server

The following database server are supported.

### MS SQL Server

Microsoft SQL Server 2016, Microsoft SQL Server 2017, Microsoft SQL Server 2019, and Microsoft SQL Server 2022.

The MSSQL databases require the `READ_COMMITTED_SNAPSHOT` set to `ON` - that is automatically taken care of by our automatic installation and our scripts.

recommended: min. Microsoft SQL Server 2019

**Important** For production not supported any SQL Express version of MS SQL Server.

## Oracle

Oracle 19 c, Oracle 18 c (18.3), Oracle 12C R2

recommended: min. Oracle 12C R2

**Important** There are special installation requirements for setting up Oracle databases prior to running the installation. For more information, refer to the *Administrator Guide*.

## PostgreSQL

PostgreSQL version 12.9 and higher

recommended: PostgreSQL version 14

**Important** You can use user name and password but no trusted connections.

## DB2

DB2 10.5 and 11.5

**Important** There are special installation requirements for setting up DB2 databases prior to running the installation.

## Azure SQL

Usage on Azure Cloud only.

## 1.2 Understanding the User Interface

---

The application is displayed in different frames: the header toolbar on the top and the main area in the middle with the fields and its snippets.

### 1.2.1 Header Toolbar

---

The header toolbar is the area on the top that shows an application name, the available working modes, as well as the signed-in user and shortcut keys.

### 1.2.2 Home Tab

---

The **Home** tab provides the **Sign in** button. Click this button to open the window to enter the user name and password. Then provide the user name and password and click **Login**.

#### User

For a Windows user, you can enter the name for a domain.

```
<domain name>\<user name>
```

#### Password

Enter the password for the provided user.

### 1.2.3 Sessions

---

Shows all current sessions of the logged in user. Open browser sessions are shown and can be deleted. The current session is explicitly indicated. All issues tokens, which are for example used by the platform's Web UIs, can also be deleted. Deleting tokens does not necessarily have an immediate effect, as there is no token revocation. Therefore, tokens will stay valid for resource services that cached them.

### 1.2.4 Management Tab

---

The **Management** tab is shown only for a signed-in administrator user. The tab shows a table of all available client IDs. If needed, you can add a new and edit or delete an existing client ID. You can also show the client ID's details in a separate window.

The table provides the following columns:

- **ClientId** - Available client Id
- **Client Secrets** - If a client secret is available for a client Id the Copy button is displayed that allows you to copy the client secret into the clipboard.
- **Permissions** - The permissions include authorization modes, introspection settings, and additional permissions.
- **RedirectUris** - a list of valid redirect uris for the client
- **PostLogoutRedirectUris** - a list of valid post logout redirect uris for the client
- **Type** - if the client is a public or confidential client.
- **Actions** - Buttons to edit, view or delete the application client registration

## 1.2.5 Roles

---

Only available for administrators.

This tab shows all available roles. These roles can be used, for example, by the platform, for internal role assignments. There are 2 actions available: deleting and hiding roles. After a role is deleted, the role is deleted from the stored list. Roles can also be hidden, which is more practical if the intention is to not make the role available for external applications, such as the platform.

Active Directory user groups are shown as a role. These roles are not the roles that are part of the platform. Commonly, a role is assigned to a platform role. Other applications can do similar assignments.

## 1.2.6 Identity Providers

---

The Authorization Server by default integrated into Windows Active Directory. The **Identity Providers** tab allows to configure settings for Active Directory. External identity providers can be configured here as well. Please refer to [How to configure identity providers](#) for more information.






In **Active Directory Settings**, you can configure whether or not AD users and groups are collected and made available in the Roles tab. You can define which groups are made available - only for the local machine, only for the domain, or both. You can specify an Active Directory name or container.

**Global IDP Settings** allow to disable the automatic collection of roles of users that are authenticated by external identity providers. The service also periodically queries all external IDPs for available roles. This interval can be set here as well.

## 1.2.7 Toolbar and Other Icons

---

The following icons are shown on the user interface.

Symbol	Name	Description
	Help	Click this icon and select either <b>Show help</b> or <b>About</b> from the context menu. Clicking <b>Show help</b> , opens the client's user guide help in a new window. Clicking <b>About</b> , opens the client's About window with copyright as well as third-party copyright information.
	Copy client ID	Click this icon to copy the client ID into the clipboard.
	Edit client ID	Click this icon to open the <b>Edit Client Id</b> window that allows you to change the current settings for the client secret, permissions, custom permissions, redirect Uris, and post logout redirect Uris.
	Show details	Click this icon to show the <b>Client ID details</b> window. The window list the client ID, client secret, the permissions, the post logout redirect Uris, and redirect Uris.
	Delete client ID	Click this icon to delete the client ID with all its details.

## 1.2.8 Shortcut Keys

The following shortcut keys are provided.

Shortcut Key	Action
Sign in	Allows the user to enter a user name and password and click <b>Login</b> to sign in to the TCG Authorization Server. The signed-in user can now access available Primus components and clients without the need to sign in again.
New Client ID	Click the button to open the <b>Add New Client Id</b> window that allows you setting up the settings for a new client ID by providing the configuration for client secret, permissions, custom permissions, redirect, and post logout redirect Uris. To save the changes, click <b>Create Client Id</b> button.
Create Client ID	Click the button to create a new client ID with the current settings in the <b>Add New Client Id</b> window.
Edit Client ID	Click the button to open the <b>Edit Client Id</b> window that allows you changing the settings for an existing client ID by providing the configuration for client secret, permissions, custom permissions, redirect, and post logout redirect Uris. To save the changes, click <b>Save Client Id</b> button.
Save Client ID	Click the button to edit a client ID with its current settings for client ID and client secret, permissions, custom permissions, redirect and post logout redirect Uris.
Sign out	Click the button and confirm the log out by pressing <b>Yes</b> to log out the current user.



## 1.3 How to get Started

---

Once the TCG Authorization Server is installed the TCG Authorization Server Web client is available.

For a normal user, the Web client provides the **Home** tab and the **Session** tab. A signed-in user can then use other applications, such as Primus Process Modeler and Process Monitor or external clients, such as, Document Review and Batch Review with single sign-on (SSO). SSO means that for the other applications you do no longer have to sign in. The **Session** tab shows all open sessions of the current user and allows to delete any issues tokens.

For an administrator user, the Web client provides several other tabs:

- **Management** tab to manage the client applications.
- **Roles** tab to list and optionally hide all roles and users that are available for role searches
- **Identity Provider** to configure Active Directory and external identity provider integrations.

For more information on how to use the user interface, refer to the [user interface](#) topic.

### 1.3.1 Creating a new Client

---

To allow single sign-on for example for an external client for Primus, you need to create a new client Id and set its configuration.

**Important** An Administrator user only can add and maintain client IDs on the Management tab.

Perform the following steps to create a new client ID.

1. From the **Management** tab click **New Client Id**. The **Add New Client Id** window is displayed.
2. Enter the client ID that uniquely identifies your application.
3. Optionally, enter the client secret. A client that has a client secret is called **confidential client**. A client secret is required when these credentials are used by a service in a client credentials grant type logon. In case of a non-confidential client like a SPA or WPF client, a client secret is not needed since it cannot be kept secret. Those clients are called **public clients**.
4. Set the Authorization modes by selecting one or more items for: **Client credentials**, **Password**, and **Authorization code**.
5. Optionally, set the introspection by selecting *Yes*. By default, the option is set to *No*. Only resource services that verify that a client is authorized to access resources require that.
6. Optionally, you can give special permissions to daemons that sign in using client credentials. Chose an option of the additional permissions. By default, *None* is selected.
  - *Authentication service admin*: useful for daemons (services) that must have the permission to automatically configure the service. For example, the platform installer uses this permission.
  - *Resource service*: gives a system claim to daemons that are resource services. Platform services have this privilege.
  - *Activity host*: gives a system claim to a daemon that is not a resource service, but that still requires system permissions. For example, Activity Server processes get this permission.
7. Add windows SIDs that the client always receives on the login. This option is only recommended for confidential clients, with the explicit use case of an Activity Server/Host that is intended to run with permissions of a technical user. In that case, the Activity Server should not get the *Activity host* additional permission.
8. Add a uri for **Redirect Uris** for clients that use the authorization code flow. Without correct redirect uris, a web client cannot use the authorization code flow as the Authorization Server refuses to redirect to unknown redirect uris. The **Redirect Uri** is case sensitive.
  - For rich client applications it is sufficient to specify `http://localhost/` to be able to use any port on localhost redirect URIs.
9. Add a uri for **Post Logout Redirect Uris**. If needed, you can add several uris as needed. The uris are case sensitive.
10. Click **Create Client Id** to create the new client Id with the configured settings. To close the window without saving any changes, click **Back to Management**.

## 1.4 Manually registering clients (applications) to the Authorization Service

---

Every application delegating authentication of users and services to the Authorization Service requires a registered client (often also called application). There exist two types of clients: public and confidential clients. Public clients are Single Page Applications or Rich Clients on a desktop, because a client secret cannot be securely hidden from a user. Confidential clients are daemon services such as the platform's configuration service. Those services run on a secured VM to that only authorized users have access and where the typical user cannot even see the configuration files.

When the platform installer is used to set up a system, it creates all clients that are required to run the platform automatically (unless you opted out of that behavior). In those cases, nothing needs to be done.

However, when you add a custom web client or new rich client, want to create a custom client for another Activity Server or need to recreate a client for a resource service of the platform, some things must be taken into consideration:

- Is the client public or confidential as in "can it keep a secret"?
- If it is a public web client, where is it hosted?
- If it is a daemon application, does it need special permissions?

**Important** When using the authorization code flow with PKCE with a custom client application (for example your website), ensure to only expect a code response and use PKCE. The Authorization Server does not support the OAuth2.0 Implicit flow.

For convenience, we provide some example scenarios that help you to pick the correct configuration for your new client registration.

**Important** When using **Redirect Uris** keep in mind that they are case sensitive. The URIs "http://my.contoso.com/editor" and "http://my.contoso.com/Editor" are not the same and lead to error. We recommend to lowercase the redirect URIs on the client side and only register lowercased redirect URIs.

### 1.4.1 Example: Create a Client for an Activity Server

---

1. Enter the **New Client ID** menu.
2. Provide a client ID as well as a client secret.
3. Select the **Client credentials** authorization mode and the additional permission **Activity host**.
4. Create the client by clicking on **Create Client ID**.

### 1.4.2 Example: Create a client for an Activity Server as technical user

---

In the past, Activity Servers had the option to run as a technical user. While a windows service can still do that, it does not have any effect on the given permissions anymore. To re-create that scenario, you need to do the following steps:

1. Enter the **New Client ID** menu.
2. Provide a client ID as well as a client secret.
3. Select the **Client credentials** authorization mode
4. Do not select **Allow introspection** nor **Additional permissions**
5. Under **Windows Security Identifiers** add the SIDs of a technical user or group that you use to configure access permissions.

## Obtaining a windows user SID

To determine the SID of a windows user, you can use the `Get-ADUser` powershell commandlets. Please see <https://docs.microsoft.com/en-us/powershell/module/activedirectory/get-aduser?view=windowsserver2022-ps> for more information.

Another approach that works via powershell is the following:

```
$username='techUser'
$user = New-Object System.Security.Principal.NTAccount($username)
$sid = $user.Translate([System.Security.Principal.SecurityIdentifier])
$sid.Value
```

### 1.4.3 Example: Create a Client to be used by a Website

SPAs are public by design since every piece of configuration is sent to an unknown browser on an unknown machine. The same is true for rich clients and may even be true for Asp.net Core MVC applications.

1. Enter the **New Client ID** menu.
2. Provide a client ID, but leave the client secret field empty. Web sites are public clients and cannot keep a secret.
3. Select the **Authorization code** authorization mode.
4. If the web app works with the platform, you must ensure enough **scopes**
  - a. Set the `interactive` scope if the client is accessing interactive activities, also known as external activities
  - b. Set the `unattended` scope if the client also must be able to access unattended activities, such as time driven, or document driven activities.
5. Provide a **Redirect Uri** and a **Post Logout Redirect Uri** that are both valid and both point to paths that your website accepts. Usually, you use an OIDC client library that handles these tasks. **Ensure** that the web site uses the redirect uris only in **lower case** when sending them to the Authorization Server - **Redirect URIs** are case sensitive. If your websites sends "https://my.site.com/oidc-login" but only registered "https://my.site.com/" or "https://my.site.com/OIDC-login", the redirect uris do not match and the login does not work.
6. Create the client by clicking on **Create Client ID**

## 1.4.4 Example: Create a Client for a Rich Desktop Application

---

Rich clients are public by design since all configuration must be on the user's local machine.

1. Enter the **New Client ID** menu.
2. Provide a client ID, but leave the client secret field empty. Rich clients are public clients and cannot keep a secret.
3. Select the **Authorization code** authorization mode.
4. If the web app works with the platform, you must ensure enough **scopes**
  - a. Set the `interactive` scope if the client is accessing interactive activities, also known as external activities
  - b. Set the `unattended` scope if the client also must be able to access unattended activities, such as time driven, or document driven activities.
5. Set the **Redirect Uri** to `http://localhost/`. Setting localhost implicitly allows any port to be used, so that a rich client can spawn a web service listening on a random free port on the local OS.
6. Leave the **Post Logout Redirect Uri** empty. This Uri must be matched explicitly and is of no use for a rich client application. **Ensure** that the rich client uses the redirect uris only in **lower case** when sending them to the Authorization Server - **Redirect URIs** are case sensitive.
7. Create the client by clicking on **Create Client ID**

## 1.4.5 Example: Create a Client for a resource service of the Primus Process Management System (a daemon application)

---

Daemon applications are run without any user interaction and thus cannot use the authorization code flow. Leaving username and password in a configuration file on disk is also not recommended. Instead, you use a clientId and clientSecret that are not directly tied to any actual user account and provide credentials necessary for a machine to authenticate itself when talking to another machine.

1. Enter the **New Client ID** menu.
2. Provide a client ID as well as a client secret.
3. Select the **Client credentials** authorization mode and the additional permission **Resource Service**.
4. Select **Allow introspection - yes**. The resource service must be able to verify access tokens it receives.
5. Create the client by clicking on **Create Client ID**.

## 1.4.6 Example: Create a Client for an application that uses the password grant flow

---

Using this grant type is strongly discouraged as the client application would always get access to the username and password. Use the **Authorization code with PKCE** flow wherever possible.

1. Enter the **New Client ID** menu.
2. Provide a client ID, but no client secret. Like the **Authorization code flow**, the **Password flow** is used by public client applications that cannot keep a secret.
3. Select the **Password** authorization mode.
4. Create the client by clicking on **Create Client ID**.

## 2. Concepts

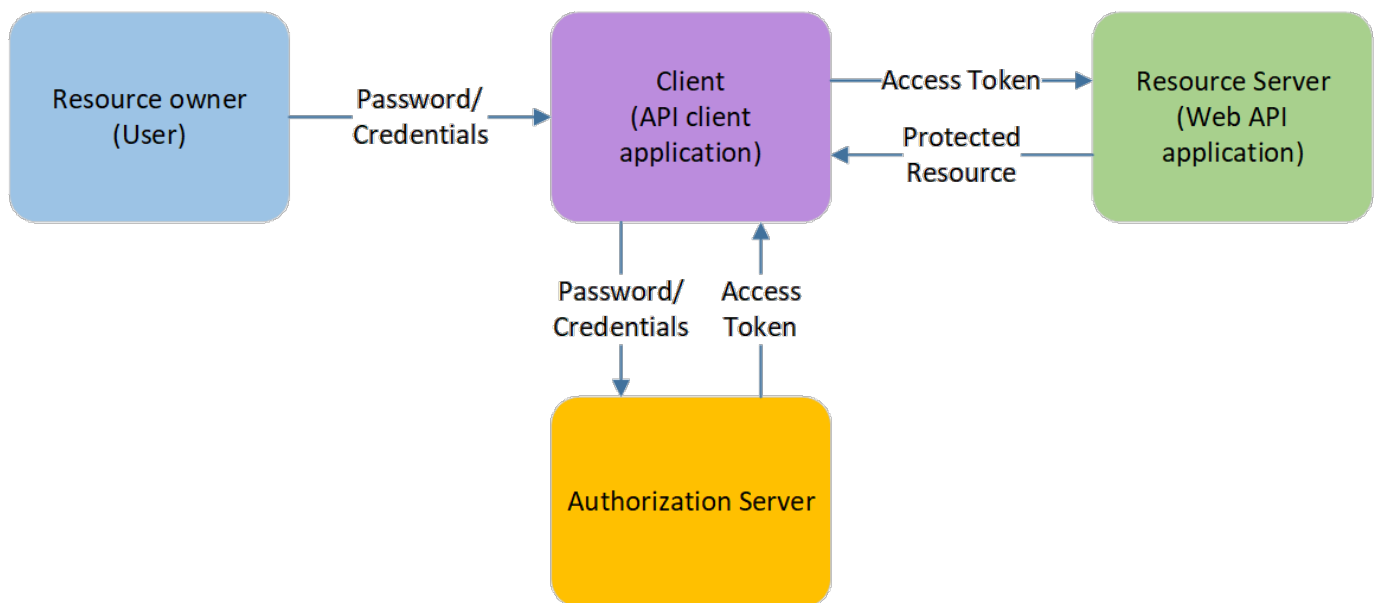
### 2.1 Platform Authorization and Authentication

Primus uses the OAuth2 protocol for the authorization of users. OAuth2 itself does not directly deal with the authentication of users and clients. However, the authentication is required to grant authorization and thus access.

The authentication provides information about “who one is”, for example, the registered Windows user *User1*. The authorization deals with the information about “who grants which permissions to whom”. For example, the authorization service grants several user rights to the Windows user *User1* based on the Windows groups to which *User1* is a member of.

Single Sign-on (SSO) occurs when a user logs in to one application and is then signed in to other applications automatically, regardless of the platform, technology, or domain the user is using. The user signs in only one time, hence the name of the feature (Single Sign-on). To achieve single sign-on (SSO) for Primus and all its components and external clients, the platform now uses OpenID Connect (OIDC). OIDC is an open authentication protocol that profiles and extends OAuth 2.0 to add an identity layer. OIDC allows clients to confirm an end user's identity using authentication by an authorization server.

For example, a user (resource owner) provides its credentials to the client, in this case the STG.RT.API, to obtain an access token from the authorization server. The access token can then be used to obtain protected resources from a resource server such as the Primus configuration service.



Note that for the sample above there are three major communication paths that are not shown in the simplifying schematic, however, the Primus uses them:

- The Authorization Server does not show the authentication service
- The Resource Server generally verifies the access token with the authorization server for validity.

In general, the access token provided by the TCG Authorization Server is accompanied with a long-lived refresh token. Once the access token expires, the refresh token is used to obtain a new access and refresh token. Each refresh token can only be used once. Refresh tokens are only intended for the client and are not intended to be passed on to a resource service. Refresh tokens provide a further layer of security

as a potential attacker can only misuse an intercepted access token for a short period of time. An access token by itself cannot be exchanged for a new access token.

Primus provides authentication via different methods:

- Authorization Code Flow with PKCE via a web browser
- User name and password authentication via the password grant flow
- Client authentication via the client credentials grant flow

### Authorization Code Flow with PKCE

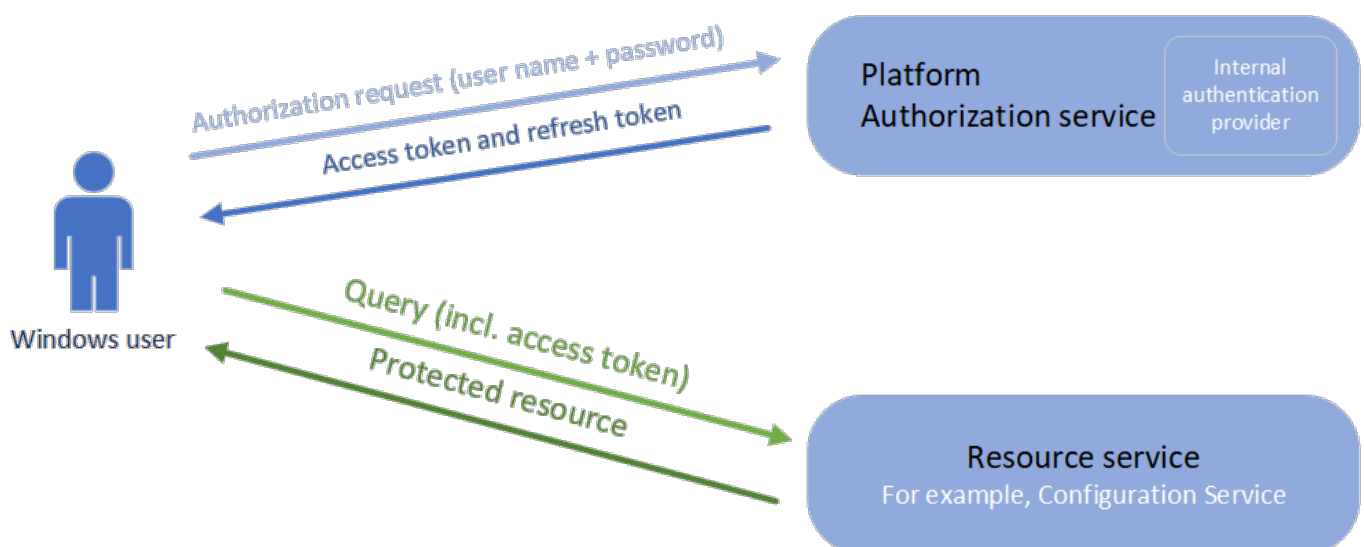
This authorization flow is the current best practice when logging into any client application because it avoids entrusting a client application with user credentials.

The **Authorization Code Flow with PKCE (proof key for code exchange)** is a security extension for public clients like web sites. A user that wants to log into a web site is redirected to the log in page of the TCG Authorization Server. If the user has not yet authenticated itself to the TCG Authorization Server, he enters a user name or password into the login field. After the credentials have been verified, the user is redirected back to the web site where he started, along with a **authorization code**. This **authorization code** is used by the website to exchange it with the Authorization Server for an access token. The **authorization code** part prevents user credentials to ever be entered into a potentially non-trusted client application. The **PKCE** part in this flow prevents the access tokens to be passed around in redirect URLs shown in the browsers URL tab, thereby preventing accidental token leakage by copy/pasting the url.

Any other website or WPF application that uses the TCG Authorization Server as its authorization authority can from that point onwards be used without having to provide the username or password again, as long as the user is still authenticated to the TCG Authorization Server via his web browser. When the user moves to a different computer or uses a different browser, he has to repeat the login flow.

### Authorization via User Name and Password

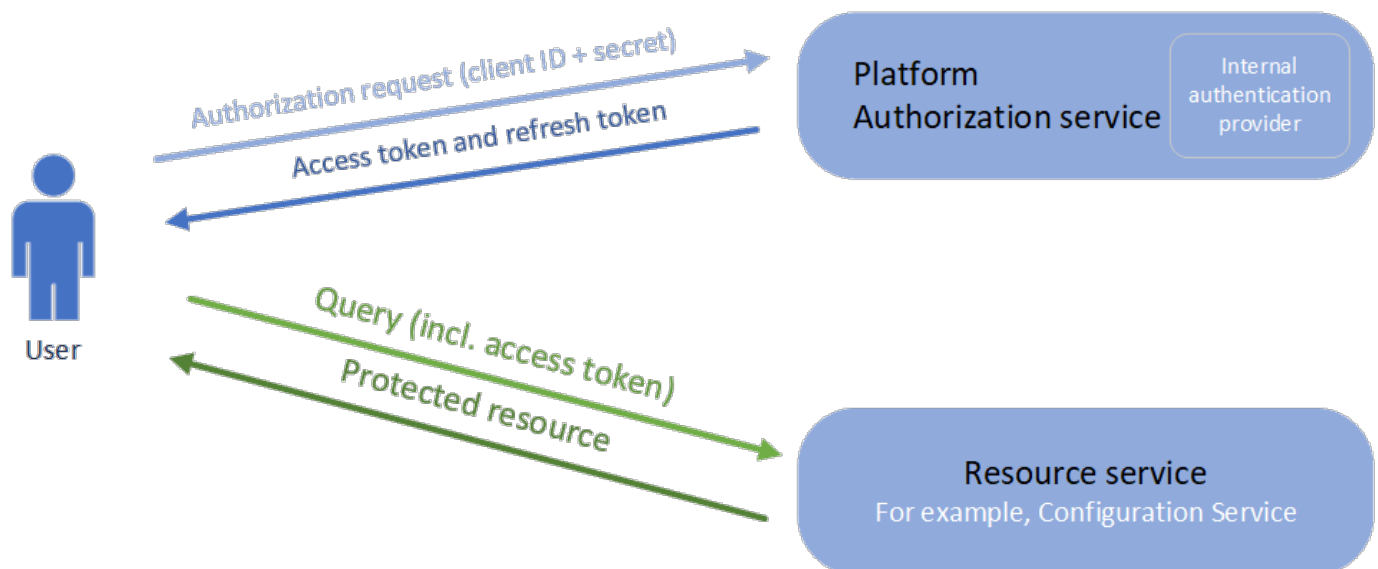
For the **password grant flow** type of the authentication, the platform sends a user name and password to the authorization service that then uses an internal authentication provider to confirm the correctness of both. Upon a correct user name and password, the authorization service creates an access token and request token with the claims of the user from the Windows groups the user is a member of.



## Authorization via client credentials

The **client credentials grant flow** type of authorization allows you to register client credentials with the authorization service, along with claims which each clientId may receive. This type of authorization is intended for machine-to-machine communication and not for typical user interactions. In this flow, a client Id and matching client secret is sent to the authorization service that issues an access token. Once the access token expires, client Id and client secret can be used again to obtain a new access token. Also, in some cases a refresh token is made available that can also be used to acquire a new access token.

Since the client credentials flow uses a client ID and secret, the administrator must make sure that these client credentials are stored in a secure manner and can only be accessed by authorized users. On Windows, they are usually stored in files where only authorized users as well as the service using these credentials has access. An additional layer of security can be added by encrypting the client secret and only giving authorized users as well as the service access to the encryption keys. The additional layer does not prevent administrators to extract the secrets from machines, but it prevents leaking the secret by accidentally copying and distributing the file containing the secret.



## 2.1.1 Refresh Token Flow

The authorization service provides an access token together with a long-lived refresh token, if the used client ID has the permission to use refresh tokens and uses the scope 'offline\_access'. Once the access token expires, the refresh token is used to obtain a new access and refresh token. Each refresh token can only be used once. Refresh tokens are only intended for the client and are never intended to be passed on to a resource service.

Refresh tokens provide a further layer of security as a potential attacker can only misuse an intercepted access token for a short period of time. An access token by itself cannot be exchanged for a new access token.





## 3. Advanced Topics

---

### 3.1 External Identity Providers

---

The Authorization Server can be configured to use an external identity provider (IDP), such as Keycloak, to authenticate users. An external IDP cannot be used to handle client credentials or custom grant types. It can only be used to handle the authorization code grant type.

For an external IDP to work, it must implement the OIDC specification and support the authorization code flow. The Authorization Server integrates into the external IDP via the .NET `Microsoft.AspNetCore.Authentication.OpenIdConnect` package, using `AddOpenIdConnect`.

Only Open ID Connect is supported. Other authentication mechanisms, such as, SAML, are not supported.

The identity provider can be configured via the TCG Authorization Server Web UI.

#### 3.1.1 Claims Required from the External IDP

---

Independent of what identity provider is used, it must provide at minimum the following claims in its ID token or via the userinfo endpoint:

- `sub`: for the unique user ID, MUST BE in both the ID token and the user info response, see OIDC specification
- `name claim`: for the user's display name, configurable.
- `username claim`: for the user's username

#### 3.1.2 Claim Required by the Platform

---

The reason why we provide a custom Authorization Server is for guaranteeing for all applications in this eco-system some authorization defaults. For example, you can theoretically use any sort of string to adorn a user with permissions to do something. Some systems provide them as a `role`, others as `roles`, yet others as `http://schemas.microsoft.com/ws/2008/06/identity/claims/role` or `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/sid`.

To relieve resource services and other applications in this eco-system from guessing and too many configuration options, some defaults must be provided among the claims of authenticated users. These defaults date back from when the platform was still tightly coupled to windows. The following defaults must be honored to ensure the relationship to the Authorization Server to work:

- A unique user ID is a claim of type `sub`
  - For backwards compatibility (3.1.0 or earlier), this claim is also present as `http://schemas.microsoft.com/ws/2008/06/identity/claims/primarysid`
  - This claim is used to remember which user locked a process for editing
- The username of the user is a claim of type `username`
  - The username is often used when assigning an entity to a particular user. We assume that the used external authentication realms do not collide in a way that the same username belongs to 2 different users.
- The display name of the user is a claim of type `name`
  - For backwards compatibility (3.1.0 or earlier), this claim is also present as `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name`
  - The name is used for displaying purpose of the user name on top of the web sites
- Roles of the user are claims of type `role`
  - For backwards compatibility (3.1.0 or earlier), this claim is also present as `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/sid`
  - The roles are used to give users access permissions, for example, to access particular processes and activities
  - To give access to only a single user, the unique user ID can be used

### 3.1.3 Configuring the Service to Use an External Identity Provider

External identity providers can comfortably be configured via the Authorization Server Web UI by any administrator. The configuration data is stored in the database and encrypted at rest with the data protection subsystem provided by .NET. The Authorization Server provides integrated support for periodically fetching roles from Entra ID and Keycloak 19 - 21.

**Important** Any change regarding external IDP configuration requires a service restart.

The external IDP must be configured to allow the Authorization Server to delegate authentication calls to it. This requires usually at minimum the following:

- A registered application or client for the external IDP. A `client ID` in OIDC terms.
- A client secret (depends on external IDP)
- A redirect URI that the external IDP will send its code response to.

The redirect URI for the Authorization Server is `https://<auth_service>/signin-oidc`, for example `https://auth.contoso.com/signin-oidc` if you host the Authorization Server on `auth.contoso.com`. Note that many external IDPs require HTTPS.

The following values must be available to be able to configure the Authorization Server:

- IDP name: used to display the sign-in button and must not be empty.
- Authority URL: a URL that is the base path of the external IDP's OIDC endpoints.
- Metadata address: only required if it is not using a standard route behind the Authority URL
- Client ID: The client ID that was created on the external IDP
- Client secret: in case a confidential client is used and you want to automatically fetch roles available on the external IDP
- User name claim type: the claim type that the external IDP writes the user name to
- Name claim type: the claim type that the external IDP writes the user's display name to
- Role claim type: the claim type that the external IDP writes the user's roles to

Additionally, you can define whether or not to

- require HTTPS endpoints for the external IDPs OIDC endpoints. It is recommend to require HTTPs endpoints.
- get additional claims from the user info endpoint, in case not all user claims are sent with the ID token
- disable automatic role discovery, in case you do not want the Authorization Server to periodically fetch and store roles that are available via the external IDP
- map custom claims, for example when you want to give certain users a specific role
- Define sign-in scopes, if the sign-in to the external IDP uses specific scopes

### 3.1.4 Configuring Claim Mappings

At the end of the configuration of the external identity provider, a list of custom claim mappings can be defined. They allow, for example, to map particular users to specific roles, or to assign a standard role to every user, substituting for the `Everyone` SID claim that users signed in via Active Directory receive.

The configuration understands 4 parameters:

- Source claim type: The claim type (for example, `role` or `preferred_username`) of the claim that the external IDP provides. Can be left empty if it defaults to the default role claim type.
- Source claim values: A space separated list of claims, used as individual regex expressions.
- Target claim type: The claim type of the target value. Can be left empty if it defaults to the default role claim type.
- Target claim value: A claim that an authenticated user that matches the source type and value receives.

In most cases, the target claim type can be omitted since this feature is generally aimed at giving users additional roles.

#### Example: Assign the role 'everyone' to every user

The following configuration assigns the role `everyone` to every user that authenticates via the external IDP.

- Leave the source and target claim types empty.
- Set the source claim value to `.*`.
- Set the target claim value to `everyone`.

### Example: Assign the role 'administrator' to specific users (Entra ID)

The following configuration assigns the role `administrator` to the users with the usernames `carl@contoso.com`, `kevin.chalet@contoso.com` and all users that are suffixed with `-admin@contoso.com` (like `charlotte-admin@contoso.com`).

- Set the source claim type to `preferred_username`.
- Set the source claim value to `carl@contoso.com .*-admin@contoso.com kevin\chalet@contoso.com`
- Set the target claim value to `administrator`

## 3.1.5 Examples

Here we provide sample configurations for the Authorization Server to add either Keycloak or Entra ID as an external IDP.

### Example: Use Microsoft Entra ID as an External Identity Provider

If you have not yet prepared Entra ID to be used as an authentication provider (for example, you have not yet registered any application), this documentation has a [dedicated section](#) that explains all required steps to set up Entra ID integration in detail.

If you already have client ID and secret, then simply follow these steps:

1. Log in as local administrator to the Authorization Server
2. Click on the "Identity Providers" tab
3. Click on "Add Entra ID"
4. Enter your configuration data. In this example, the following data is sufficient:

```
json
{
  "Name": "EntraID",
  "Authority": "https://login.microsoftonline.com/<tenant_guid>/v2.0",
  // MetadataAddress is not required since it uses the default path behind the Authority URL
  // "MetadataAddress": "https://login.microsoftonline.com/<tenant_guid>/v2.0/.well-known/openid-configuration",
  "ClientId": "<registered application (client) id>",
  "ClientSecret": "<registered application secret>",
  "UsernameClaimType": "preferred_username",
  "RoleClaimType": "roles",
  "NameClaimType": "name",
}
```

5. Optionally add custom claim mappings
6. Click "Create" and restart the Authorization Server

### Example: Use Keycloak as an External Identity Provider

This example below uses a local Keycloak installation on Docker. In real scenarios, the URL is not localhost and uses https.

1. Log in as local administrator to the Authorization Server
2. Click on the "Identity Providers" tab
3. Click on "Add Keycloak"
4. Enter your configuration data. In this example, the following data is sufficient:

```
json
{
  "Name": "Keycloak",
  "Authority": "http://localhost:8080/realms/master",
  // MetadataAddress is not required since it uses the default path behind the Authority URL
  // "MetadataAddress": "http://localhost:8080/realms/master/.well-known/openid-configuration",
  "ClientId": "oidc-code-pkce",
  "ClientSecret": "87pdXsj0O749Z2s9hO3iq3c3TydcF6vJ",
  "UsernameClaimType": "preferred_username",
  "RoleClaimType": "role",
  "NameClaimType": "name",
  "RoleDiscoveryBaseUrl": "http://localhost:8080/admin/realms/master/"
}
```

5. Optionally add custom claim mappings
6. Click "Create" and restart the Authorization Server

### Troubleshooting Permissions or Claims Given By External IDPs

As explained at the beginning of this chapter, it is important that the external IDP provides the required claims. If the external IDP, for example, does include roles in the ID token or via the user info endpoint, the authenticated user also does not have any roles. The Authorization Server shows you the claims of the current user on the main page if you set the `appsettings.production.json` setting `Service:ShowClaimsOnHome` to `true`. Knowing this list of claims helps to fine-tune the integration into the external IDP.

Keycloak, for example, does not send any roles with a freshly pulled up docker container (`quay.io/keycloak/keycloak:20.0.1`). To get roles for a user authenticated by Keycloak, you have to change Keycloak configuration first.

1. Click on `Client Scopes`, then edit the `roles` scope.
2. Click on `Mappers`, add a mapper `by configuration`, select the `User Client Role` mapper type.
3. Choose a name, leave the `Client ID` empty and set the `Token Claim Name` to `role` or `roles`, and ensure `Add to ID Token` is selected.
4. Repeat the login into the Authorization Server, your user now has claims derived from his Keycloak roles.

For further finetuning and Keycloak configuration, we have to defer you to Keycloak documentation or to a Keycloak consultant of your choice.

### **Troubleshooting redirect URIs**

The Authorization Service sends a redirect URL to the external IDP so that users can perform the authorization code flow with PKCE. The redirect URIs are usually case sensitive, but that is ultimately decided by the IDP, so results may differ. Read up on the requirements of your external IDP and verify that the redirect URI matches the exact casing.

## 3.2 Using Entra ID as an External Identity Provider

---

This section explains the steps required to use Entra ID as an external IDP. It starts by describing the configuration on the Azure Portal, and then describes the configuration settings of the Authorization Service.

### 3.2.1 Requirements

For this to work you need an Azure subscription that provides you with the Microsoft Entra ID service and an Entry ID tenant. For minimal requirements, the basic account of Entra ID is sufficient. Having a paid subscription can make the role assignments to users easier. You also need permissions on the Azure Portal to register an application, as well as the permission to add an enterprise application.

### 3.2.2 Preparations in the Azure Portal

Microsoft keeps an own guide (<https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/add-application-portal>) on how to add applications to Entra ID. If the UI changes, the Microsoft guide is likely more up to date than this documentation. The Microsoft guide also offers more detail than what we can provide.

The steps you need to perform on the Azure Portal provide you with all the configuration settings to connect the Authorization Service to Entra ID. Those settings include:

- a client ID
- a client secret
- the tenant guid that identifies your organization on Entra ID

#### Create a new Application

1. Open the Microsoft Entra ID management blade on the Azure Portal
2. Click on 'App Registrations'.
3. Add a 'New Registration'.
  - Define a display name
  - Allow accounts in this organization. If you're familiar with Entra ID, you can of course also chose another option
  - Select 'Web' on the 'Redirect URI' part
  - Define a default **redirect URI**  
 The URI points to the Authorization Service so that if you host on `https://auth.contoso.com` you have to specify `https://auth.contoso.com/signin-oidc`.  
 Use only lower-case characters - redirect URIs are case-sensitive.
4. You are now redirected to the 'Overview' page of the created application. The applications 'ClientID' is shown under 'Essentials'.
5. Click the 'Register' at the bottom.
6. In your newly created application, click on 'Add certificate or secret' to add a client secret that the Authorization Service requires for integrating into Entra ID. You also reach this area by clicking on the 'Certificates & Secrets' area in the 'Manage' group on the left.

7. Click on 'New Client Secret' - an area opens on the right.

- Enter a description
- Select a reasonable expiration date of the client secret. Keep in mind that you need to update the secret configured for the Authorization Service after the secret expired.

8. Keep a copy of the value of your secret - it cannot be viewed after you leave this page.

After these steps you have 4 important values:

1. You created an application, and by doing that you obtained a **client ID**
2. The application showed you also the **tenant guid**
3. You created a **client secret** that allows the Authorization Service to use the client credentials flow
4. You defined a correct, case-sensitive **redirect URI** that points to the authorization service.

**Tip** Redirect URIs are a common pitfall. The RFC specification defines some basic requirements, but each IDP can have additional constraints. If the redirect URI is an issue, please refer to the Microsoft documentation for redirect URI restrictions and limitations (<https://learn.microsoft.com/en-us/entra/identity-platform/reply-url>).

If you do not want to define and assign roles to users, you can jump to the configuration of the Authorization Service.

### Add Application Permissions for Role and User Discovery

The Authorization Server can actively discover roles and users that are available for signed-in Entra ID users. This search uses Microsoft Graph and requires additional permissions. The permissions are set on the client application.

1. Navigate to the client application that you just created.
2. Click on 'API permissions' in the menu on the left.
3. Click 'Add a permission'. A menu on the right opens.
  - a. Click on Microsoft Graph
  - b. Click on 'Delegated permissions'
    - i. Select the delegated 'User.Read' permission
  - c. Click on 'Application permissions'
    - i. Select the application 'User.Read.All' permission
  - d. Click the button 'Add permissions' at the bottom of this panel.
4. Some permissions might require *Admin Consent*. If the consent is missing, you see a yellow warning icon.
  - a. Switch over to 'Enterprise applications' - there is a link below the table.
  - b. Click the big button 'Grant admin consent for XX' and grant admin consent.

If these permissions are not given, automatic role and user discovery of the Authorization Server does not work. Information-lvl logs are written if Microsoft Graph responds missing permissions or other issues.

### Define Roles and Apply them to Users

Application roles can be used to group your users. The roles can then be used by platforms and applications that authenticate users via the Authorization Service to assign permissions in their internal role management. Assigning roles/groups instead of users individually allows you easier configuration than assigning each user individually.



To create application roles in Entra ID, you must first add roles to your application and then assign users to these roles. Here, we explain the steps needed to be done in the Azure portal. However, it is likely possible to use scripts and the Azure command line interface to automate that process. Refer to the Azure documentation on how to do so.

Follow these steps to manually create an application role and assign a user to that role:

1. Click on 'App Roles' and create a new role. Chose any name, for example 'users' or 'administrators'
  - Repeat this step for all roles that you want to add
2. Click on 'How do I assign App roles' - a new properties panel opens on the right. Click on the link to 'Enterprise applications' that redirects you to the configuration of your application in Enterprise application management
3. Click 'Users and Groups' on the 'Management' options group, or click the first button 'Assign users and groups' in the 'Getting Started' section in the Overview
4. Assign users to roles
  - Click the button 'Add user/group' in the top menu.
  - Select one or more users you want to assign to a role
  - Select the role you want to assign to those users. If you have no roles defined yet, please define a role in your application first
  - Click on 'Assign' at the bottom. You now have assigned one role to the selected users
  - Repeat this step for all other roles that you want to assign to users

Creating roles is done in 'Application Registration', while assigning users is done in 'Enterprise applications'. Navigating normally between both is slow. We recommend using the respective shortcuts: 'How do I assign App roles' -> 'Enterprise Applications' on your applications 'App Roles' section to switch to the Enterprise application section of your app, and the link to 'application registration' on the 'Users and Groups' configuration area in the Enterprise Application panel of your app.

### 3.2.3 Configuring the Authorization Service for Entra ID integration

The final configuration is done entirely in the Web UI of the TCG Authorization Server. The data is stored and encrypted at rest in the database. The encryption uses the data protection subsystem of .NET.

1. Log in as local administrator to the Authorization Server
2. Click on the "Identity Providers" tab
3. Click on "Add Entra ID"
4. Enter your configuration data. In this example, the following data is sufficient:

```
{
  "Name": "EntraID",
  "Authority": "https://login.microsoftonline.com/<tenant_guid>/v2.0",
  // MetadataAddress is not required since it uses the default path behind the Authority URL
  // "MetadataAddress": "https://login.microsoftonline.com/<tenant_guid>/v2.0/.well-known/openid-configuration",
  "ClientId": "<registered application (client) id>",
  "ClientSecret": "<registered application secret>",
  "UsernameClaimType": "preferred_username",
  "RoleClaimType": "roles",
  "NameClaimType": "name",
}
```

5. Optionally add custom claim mappings
6. Click "Create" and restart the Authorization Server

## 3.3 Writing Plugins for the Authorization Server

---

You can write custom plugins for the Authorization Server to:

- decide how users are verified via username + password: `IUserLogin`
- handle custom credential flows aka custom grant types: `ICustomGrantSignIn`
- add custom role discovery for a non-integrated identity provider: `IRoleDiscovery`
- to modify claims given by an external identity provider: `IExternalIdpClaimsMapper`
- to add custom services that your plugins might need (with reservations - if you remove required or add conflicting services, things can break)

A nuget package is provided, called `AuthorizationService.Interface`. If it was not part of the delivered product, contact your vendor.

### 3.3.1 Conventions

---

A plugin is written to adjust or define users claims, or to let the know all available roles.

The following claims are required:

- sub: each identity must have a subject claim that has the type name 'sub'
- username: the username of the user, having the claim type 'username'
- name claim: the ClaimsIdentity must have a name claim of the type of the identity's name claim type
- role claims: roles of the ClaimsIdentity are only taken into account if they have the type of the identity's role claim type

Sample implementations are made available in the `AuthorizationService.PluginSample` project.

- `JsonFileUserLogin` shows how to implement the `IUserLogin` and which claims are expected
- `CustomEntraIDClaimMapper` shows how claims coming from an IDP can be adjusted. Of course, Entra ID is supported implicitly so that a plugin for that is not required.
- `CustomRoleDiscovery` shows in a crude example how roles are collected via a web request. Usually, authenticated requests are required for that. Doing so, is the responsibility of the implementer.

### 3.3.2 Implementation

---

To write a plugin, create a new .net project and reference the `AuthorizationService.Interface.dll` distributed with the Authorization Server. Implement a class with a parameterless constructor for the interface `IAuthorizationServerPlugin`. Ensure that there exists exactly **one** implementation of this interface. Plugin DLLs providing multiple or zero implementations result in service failure.

There are code samples in the `AuthorizationService.PluginSample` project

By default, the Authorization Server provides services for `IUserLogin` and `IExternalIdpClaimsMapper`. The default user login works with Windows Active Directory only. Default services also exist for `IRoleDiscovery` aimed at Active Directory, Entra ID and Keycloak

(at least version 19, 20, 21). An additional services automatically scrapes new roles from each authenticated user. There is no default for `ICustomGrantSignIn`.

```
public class ExampleAuthorizationServerPlugin : IAuthorizationServerPlugin
{
    /// <summary>
    /// The method configures authorization server to use custom classes.
    /// Useful methods are for example:<br/>
    /// <see cref="IUserLogin"/>, <see cref="ICustomGrantSignIn"/> as singletons.<br/>
    /// When using an external IDP (like Entra ID or Keycloak), you can plug into adjusting claims as well as providing role discovery
    via<br/>
    /// <see cref="IExternalIdpClaimsMapper"/> and <see cref="IRoleDiscovery"/> as transient services.<br/>
    /// </summary>
    /// <remarks>
    /// Be careful about adding additional services - you might be replacing or breaking required functionality.
    /// </remarks>
    public void SetupServices(IServiceCollection services, IConfiguration configuration)
    {
        // replaces the usual windows login
        services.TryAddSingleton<IUserLogin, CustomUserLogin>();
        services.TryAddSingleton<ICustomGrantSignIn, OurCustomGrant>();
        // adds another role discovery service
        services.AddTransient<IRoleDiscovery, CustomRoleDiscovery>();
        // adds another claims mapper
        services.AddTransient<IExternalIdpClaimsMapper, CustomClaimsMapper>();
    }
}
```

- **IUserLogin**: Verifies password credentials and implements support for custom grant flows.
- **ICustomGrantSignIn**: Receives custom request data that is transformed into a `ClaimsPrincipal`.
- **IExternalIdpClaimsMapper**: Allows to define explicitly how claims of an external IDP have to be mapped onto the authenticated `ClaimsPrincipal`.
- **IRoleDiscovery**: periodically executed and returns a list of all available roles that users may get. Only of interest if the identity provider is not supported implicitly.

**Tip** While developing a custom plugin it may be hard to determine the claims the authenticated principal receives. The setting `Service->ShowClaimsOnHome` enables displaying claims of the principal. It shows 2 sets: the claims that the user has on Authorization Server and the claims that the user has assigned to issued the access token. The `IExternalIdpClaimsMapper` adjusts the claims of the user for Authorization Server.

To configure the Authorization Server to use a plugin DLL, adjust manually or via a custom scripting task the `appsettings.production.json` found in the directory of the Authorization Server. If the file does not yet exist, you can safely create it and add the required settings. When using the platform installer, this section in the settings is not overwritten.

```
{
  "Service": {
    "PluginAssemblyPath": "c:\\path\\to\\your\\plugin.dll"
  }
}
```

## 3.4 Certificates

By default, when setting `Authorization.AutoCreateCertificates` to `false` or leaving it empty, the Authorization Server uses default certificates of the user that runs the Authorization Server. The keys that are used for .NET DataProtection are stored in `%localappdata%\ASP.NET\DataProtection-Keys` where they are encrypted at rest by the Windows DPAPI. This is the easiest approach and requires the least amount of manual configuration.

Which user runs the Authorization Server process (this becomes important at the end of this section)? When the Authorization Server runs on an application pool with the `ApplicationPoolIdentity`, its user is `IIS APPPOOL\<appPoolName>`, e.g. `IIS APPPOOL\DefaultAppPool`. When the application pool runs as a technical user, e.g. "techie", then the user identity is `<machine_name>\techie`. During startup, the Authorization Server logs the user identity it is using, in case it is unclear.

We recommend to store the .NET DataProtection keys in the database. For security reasons, these keys must be protected at rest. This is ensured by configuring certificates that are used by the data protection system.

The DataProtection keys are stored in the database, when the app setting `Service.DataProtection.PersistKeysInDatabase` is set to `true`.

### 3.4.1 Installing Certificates

In general, we recommend to use the Primus installer to set up certificates.

When certificates are changed later, users that already logged in have to log in again. Also, all client secrets cannot be decrypted anymore. Any created client secret will continue to work though.

The Authorization Server contains a cmd line call that automatically creates certificates and adjusts the `appsettings.production.json` file for the Authorization Server. Changing the certificates requires a restart of the Authorization Server, usually done by restarting the application pool when hosting on IIS.

```
AuthorizationService.exe setup certificates -s LocalMachine -j .\appsettings.Production.json
```

creates 5 certificates in the LocalMachine certificate store. To run multiple Authorization Server service instances, they must use the same certificates. The following certificates are created:

- Auth DataProtection Key Protector Certificate - protects the .NET DataProtection keys that are stored in the database
- 2x Auth DataProtection Key Unlock Certificate - allows for certificate rotation, used by the .NET DataProtection subsystem
- Auth OpenIddict Server Encryption Certificate - for encrypting data that the OIDC framework OpenIddict creates
- Auth OpenIddict Server Signing Certificate - for signing purposes of the OIDC framework OpenIddict

By default, the Authorization Server assumes to run as a single service. It can, however, be installed on multiple machines. If those machines are reachable by different URLs, they do not serve the purpose of SSO, as the usual SSO pattern works by using a browser cookie to keep the user signed in. However, for machine to machine communication that does not rely on SSO, this can reduce load on the primary Authorization Server.

**IMPORTANT:** Please remember the part at the beginning of this section where the actual service user identity was mentioned. This is important because certificates are usually not accessible to anyone on a system due to being sensitive data. When using certificates from the certificate stores, you must ensure that the app pool's user identity has access to the certificate, **including** the private key. Otherwise, the certificate cannot be used to decrypt encrypted data.

You can manually edit and assign permission for certificates stored in the local machine certificate store. Open the local computer certificate store mmc (select Run from the Start menu, and then enter certlm.msc), then use the steps `select certificate -> right mouse click -> All Tasks -> Manage Private Keys...`. For the `IIS APPPOOL\DefaultAppPool`, assign the `IIS_IUSRS` group, for technical app pool users assign that technical user or a user group of that user to the private key permissions.

### 3.4.2 Transferring certificates between machines or reusing existing certificates

---

The Primus installer does not offer an option to configure the certificate thumbprints directly. It also does not have options to transfer certificates automatically between machines.

Please create your own powershell scripts to automate this task, or use the manual steps described earlier.

## 3.5 Application Settings explained in detail

---

This chapter explains the following configuration sections:

```
{
  "Serilog" : {},
  "ConnectionStrings" : {},
  "Service" : {},
  "AllowedHosts": "*"
}
```

### 3.5.1 Serilog

---

The serilog settings are documented well here: <https://github.com/serilog/serilog-settings-configuration>

The Authorization Server ships, by default, with the sinks file, eventlog, async and seq, as well as the log enrichers

- FromLogContext
- WithMachineName
- WithThreadId
- WithProcessId

### 3.5.2 ConnectionStrings

---

The connection string section consists of key-value pairs of database provider to connection string. The database provider that is used by the application is set in `Service.DatabaseProvider`.

The following data providers are supported: `mssql`, `postgresql`, `oracle`, `db2`, `sqlite` and `inmemory`. The last two providers are only recommended for demo or development purposes.

```
"ConnectionStrings": {
  "postgresql":
"Server=127.0.0.1;Port=5432;Database=auth;Userid=postgres;Password=postgres;Pooling=false;MinPoolSize=1;MaxPoolSize=20;Timeout=15;SslMode=
"oracle": "Data Source=localhost:1521/orcl.docker.internal;User Id=\"C##test_auth_user\";Password=password",
"mssql": "Data Source=localhost;Initial Catalog=auth;Integrated Security=True",
"sqlite": "Data Source=c:\\data\\auth.db"
}
```

### 3.5.3 Service

This is the main configuration section of the Authorization Server.

- **DisableHttpsRequirement**: whether or not the Authorization Server will answer non-https requests. Defaults to `false`. This should only be enabled if it can be secured that no http traffic is susceptible for interception
- **Ssl**: whether or not to use HSTS and HTTPS redirects
- **Session**: fine-tuning SSO session parameters, allowing to trade ease of use with tighter security
- **DatabaseProvider**: defines the connection string that is used and the database provider that will connect to a DB backend
- **DataProtection**: configures the .NET DataProtection subsystem. In simple scenarios with only 1 Authorization Server it is not necessary to adjust it at all
- **AuthCertificates**: encryption and signing certificates used by the OIDC framework OpenIddict
- **PluginAssemblyPath**: allows to replace the default user authentication mechanism that works against the local Windows machine and/or ActiveDirectory with a custom authentication mechanism
- **Cors**: allows to set allowed request origins. If set, only provided origins are allowed to send requests
- **Proxy**: see "Setting up a load balancer" in the installation guide for more information

A short sample of this section

```
{
  "Service": {
    "DisableHttpsRequirement": false,
    "Ssl": {
      "EnableHttpsRedirection": false,
      "UseHsts": true,
      "HstsMaxAgeHours": 720
    },
    "Session": {
      "ExpirationTimeSpanSeconds": 7200, // expires the cookie after 7200 seconds. By default, the cookie expires only after 14 days.
      "CookiePath": "/auth", // in case of co-hosting with other services on a single VM, you can limit when the cookie is sent with requests. In this case, the AuthorizationServer is hosted on https://my.contoso.com/auth
      "RefreshTokenLifetimeSeconds": 7200, // issued refresh tokens expire after 7200 seconds. The default is 14 days.
      "AccessTokenLifetimeSeconds": 1200 // shortlived access token. Default is 1 hour
    },
    "DatabaseProvider": "mssql",
    "DataProtection": {
      "ApplicationName": "myapp",
    },
    "AuthCertificates": {
      "Style": "automatic"
    },
    "PluginAssemblyPath": null, // "C:\\path_to\\AuthorizationService.SamplePlugin.dll",
    "Cors": {
      "Origins": []
    },
    "Proxy": {}
  }
}
```

#### DataProtection

The .NET DataProtection subsystem is used to ensure encryption at rest of sensitive data such as client secrets or user tokens.

If the keys are not stored in their default location, either by using `PersistKeysInDatabase` or `PersistKeysPath`, they are not encrypted at rest unless they are explicitly protected by configured certificates. Therefore, we recommend to set `ProtectKeysThumbprint` to the thumbprint of

a certificate that can be found in the Windows certificate store that is accessible by the Authorization Server user. If that is not possible, provide the file and password.

- **ApplicationName**: if set, allows shared access to protected payloads (data protection) if application base paths differ but multiple installed apps are working together
- **PersistKeysInDatabase**: if set to `true`, the keys of the .NET DataProtection subsystem are stored in the Authorization Server DB. Defaults to `false`.
- **PersistKeysPath**: in production scenarios with multiple machines, it is recommended to persist keys to a path and protect them. Defaults to `""`.

```
{
  "DataProtection": {
    "ApplicationName": null,
    "PersistKeysInDatabase": false,
    "PersistKeysPath": "",
    "ProtectKeysThumbprint": "",
    "ProtectKeysFile": "",
    "ProtectKeysPassword": "",
    "UnprotectWithMultipleKeys": false,
    "UnprotectMultipleThumbprint1": "",
    "UnprotectMultipleThumbprint2": "",
    "UnprotectMultipleFile1": "",
    "UnprotectMultiplePassword1": "",
    "UnprotectMultipleFile2": "",
    "UnprotectMultiplePassword2": ""
  }
}
```

## AuthCertificates

These are certificates used by the OIDC framework OpenIddict. They may be used for encryption and signing of tokens. By default, OpenIddict automatically creates "development" certificates that are perfectly functional for a simple installation.

- **Style**: defaults to `automatic` that automatically sets up certificates for the service user. For distributed installations, `persisted` is recommended
- **EncryptionCertificates/SigningCertificates**: lists of certificates that are used for encryption and or signing. Multiple certificates can be specified to support certificate rotation
- **Thumbprint**: A certificate thumbprint either in the LocalMachine or CurrentUser personal store (`my`). The Authorization Server requires of course access
- **Path/Password**: in case it is not possible to use certificates from the certificate stores. Since the password cannot be protected, this requires that that machine on that the Authorization Server runs is secured.

`/"StyleOptions": "ephemeral (default), automatic, persisted",`

```
{
  "AuthCertificates": {
    "Style": "persisted",
    "EncryptionCertificates": [
      {
        "Thumbprint": "a5511be2236f5d7f6ec8d95dc7e37fc54bf1e14b"
      },
      {
        "Path": "c:\\auth_svc\\openIddict_enc_cert2.pfx",
        "Password": "my secret pwd"
      }
    ],
    "SigningCertificates": [
      {
        "Thumbprint": "a5511be2236f5d7f6ec8d95dc7e37fc54bf1e14c"
      }
    ]
  }
}
```



```
}  
}
```

## 3.6 Using Postman to Test Authentication Flows

In some cases it helps to look at the authorization calls directly to verify that an installation is working properly or to troubleshoot problems. We will showcase how to use Postman (<https://www.postman.com/>) to send web requests to the TCG Authorization Server to test authentication and authorization calls. Other API clients such as Insomnia will work similarly (<https://insomnia.rest/>).

This section is intended for developers or person familiar with HTTP. Experience with API clients such as Postman is highly recommended.

For the following samples we assume that the authorization service is hosted on <https://auth.contoso.com>.

### 3.6.1 Requesting the Open ID Connect discovery document

The discovery document contains Open ID Connect configuration and is used by many client libraries to configure the endpoints that will be used to request authorization, tokens or certificates.

1. Create a new request
2. Use the HTTP GET method, set the URL to <https://auth.contoso.com/.well-known/openid-configuration>

The response will be the discovery document. You can of course also use a browser and just paste a url adjusted to your authorization service host.

```
{
  "issuer": "https://auth.contoso.com/",
  "authorization_endpoint": "https://auth.contoso.com/connect/authorize",
  "token_endpoint": "https://auth.contoso.com/connect/token",
  ...
}
```

### 3.6.2 Obtaining an Token via Grant Type Client Credentials

Client credentials are used by the Primuss resource services. They expect that on the Authorization Server a confidential client is configured, along with the permissions **Interactive Activities**, **Unattended Activities** and **Process Monitor**.

1. Create a new request
2. Use the http POST method, set the URL to <https://auth.contoso.com/connect/token>
3. Do not use parameters, leave the Authorization as **No Auth**
4. The body is x-www-form-urlencoded
5. Set the following values:
  - a. client\_id: your\_confidential\_client\_id
  - b. client\_secret: your\_client\_secret
  - c. grant\_type: client\_credentials
  - d. scope: openid interactive unattended procmon

The response, if successful, looks like

```
{
  "access_token": "8fpSK01eKQf01sz0FcFDUZW17QZettDmu-j2CuXzwK4",
```

```

"token_type": "Bearer",
"expires_in": 3599,
"id_token": "<a signed JWT>"
}

```

### 3.6.3 Obtaining a Token via the Authorization Code Flow

The authorization code flow is a 2-step procedure that makes it possible that the web client that requires access to a resource can be authorized to retrieve an access token. The authorization happens by the user authenticating to the Authorization Server, potentially entering his user credentials.

By using this flow, the web client does not ever get access to user credentials, making this the current de-facto standard for authenticating users across the web. The authorization code flow requires a code challenge that depends on calculating the hash (usually SHA256) of a client-side generate code\_verifier.

To test this flow, you need a registered client for the **Authorization Code** flow, with any redirect URI (in this example we use `http://localhost/signin-oidc`) and the permission **Interactive Activities**.

1. Create a new GET request to `https://auth.contoso.com/connect/userinfo`
2. Go to the Authorization Tab below the URL input
3. Select Type **OAuth 2.0**
4. In **Configure New Token**, configure:
  - a. Grant Type: **Authorization Code (With PKCE)**
  - b. Callback URL: `https://localhost/signin-oidc` (but do not select *Authorize using browser*)
  - c. Auth URL: `https://auth.contoso.com/connect/authorize`
  - d. Access Token URL: `https://auth.contoso.com/connect/token`
  - e. Client ID: `your_public_client_id`
  - f. Client Secret: *leave empty*
  - g. Code Challenge Method: `SHA-256`
  - h. Scope: `offline_access openid interactive profile`
  - i. State: `1234`
5. Click **Get New Access Token**
6. A dialog opens and expects you to log into the Authorization Server. Log in.
7. The *Token Details* show your access token, your id token (due to scope *openid*) and your refresh token (due to scope *offline\_access*)
  - a. copy your refresh token.
8. Click **Use Token**

Now send the GET request to the userinfo endpoint and retrieve information about the roles and scopes that are applied to your user.

### 3.6.4 Obtaining a Token via Grant Type Refresh Token

If you do not yet have a refresh token, perform the steps in the [Authorization Code Flow](#) example and copy the refresh token once your login was successful.

1. Create a new POST request to `https://auth.contoso.com/connect/token`
2. Go to the **Body** tab, select x-www-form-urlencoded
3. Enter the following keys and values:
  - a. `client_id`: your\_public\_client\_id
  - b. `grant_type`: refresh\_token
  - c. `refresh_token`: YY4DaZOBuFXw3nZlRYog1WVmb3eKMJeLIL2GmqbmBrQ (actually, the refresh\_token you copied earlier)
4. Send the request

The response contains a new access and refresh token, tells you the new token expiration (in 60 minutes), the granted scopes and the id token with user information that is used by the client application to display information about you.

```
{
  "access_token": "y3tNjPxx6KZaWtumRZnYSzQqex_WELVBfxV4tM-WkYE",
  "token_type": "Bearer",
  "expires_in": 3600,
  "scope": "openid offline_access interactive profile",
  "id_token": "<a signed JWT>",
  "refresh_token": "HWj-2ejYlFlU5HjKZagY0GTe_XJZ5JoTzqZ2yuqNb8k"
}
```

**Important** The issuer of the ID token must be verified and must match the authority URL.

## 4. About

---

### 4.1 Release Notes

---

The TCG Authorization Server release notes contain important information that you are recommended to read before using this product.

#### 4.1.1 What's new for Version 3.1.3

---

- The routes `api/roles`, `api/users` and `api/identityproviders` are now available for confidential clients with the resource service permission. Querying roles is used for user role assignments, for example on the platform service.
- Support for custom schema names on PostgreSQL, as long as they are lowercase. Also see the *Installation Configuration* section in the Installation Guide.
- On Oracle, an unsupported transaction isolation level was logged as an error by the Oracle library. Database access code was changed to not use that isolation level with oracle to prevent the logged error. The error did not inhibit functionality.
- The database configuration for SQLServer has been extended to support mandatory and strict SSL connections to the SQLServer. Two new properties have been added to the configuration.  
For backwards compatibility, in case encryption has not been specified, the service defaults to trust the certificate chain and to only optionally use encrypted communication.
- Azure AD (Azure ActiveDirectory) has been renamed to (Microsoft) Entra ID
- Entra ID user discovery gets paginated responses and did not discover more than 100 users. The new implementation fetches all available pages and enlarges the page size to 999.
- Software Bills of Material are now part of the installation. They are found in the installation directory under "SBOM":  
`<install_dir>\SBOM.`

#### Updated Vulnerable Dependencies

Notable are:

- System.IdentityModel.Tokens.Jwt 6.24.0 -> 6.35.0 (moderate):  
Microsoft Security Advisory CVE-2024-21319: .NET Denial of Service Vulnerability
- Microsoft.Data.SqlClient 2.1.4 -> 5.1.4 (high):  
Microsoft.Data.SqlClient and System.Data.SqlClient vulnerable to SQL Data Provider Security Feature Bypass

#### 4.1.2 What's new for Version 3.1.2

---

- Add an API endpoint that allows searching for users by username (GET `~/api/users/?name=JohnDoe`)
- Add an API endpoint to get the current version (GET `~/api/version`)
- Add an API endpoint for get active identity provides, for example, ActiveDirectory and Keycloak (GET `~/api/identityproviders`). Administrative permissions are required.
- ID Tokens now contain `authority` and `identity_provider` entries to expose how the user was authenticated.

## Fixed Thirdparty Vulnerabilities

- Update Oracle.EntityFrameworkCore to version 6.21.90 due to CVE-2023-21893 in Oracle.EntityFrameworkCore <https://github.com/advisories/GHSA-5pm2-9mr2-3frq>: Component takeover in Oracle Data Provider for .NET
- Update System.Security.Cryptography.Pkcs from version 6.0.1 to 6.0.3 due to CVE-2023-29331 <https://github.com/advisories/GHSA-555c-2p6r-68mm>: .NET Denial of Service vulnerability

## Bugfixes

- The external IDP configuration panel did not allow setting multiple claim mappings, or to delete claim mappings. Disabling the password login box when using an external identity provider is fixed.
- IIS request filtering is set to 10000kb by default, to prevent issues with large ID tokens. Extend documentation about IIS request filtering limitation and how to configure scopes for rich client application registrations
- Provide a more helpful error message when configuring an invalid redirect URI (fragments are not allowed)
- The auth-admin client registration incorrectly received password and authorization code flow permissions. It now only has the client credential authorization flow enabled.

## Bugfixes

- The external IDP configuration panel did not allow setting multiple claim mappings, or to delete claim mappings. Disabling the password login box when using an external identity provider is fixed.
- IIS request filtering is set to 10000kb by default, to prevent issues with large ID tokens. Extend documentation about IIS request filtering limitation and how to configure scopes for rich client application registrations
- Provide a more helpful error message when configuring an invalid redirect URI (fragments are not allowed)
- The auth-admin client registration incorrectly received password and authorization code flow permissions. It now only has the client credential authorization flow enabled.

## 4.1.3 What's new for Version 3.1.1

Version 3.1.1 contains security improvements, better external Identity Provider integration, and several bug fixes.

- Security for the user has been improved:
  - Session cookies allow for deleting of a session on the server side.  
After a sign-out, even using a copy of the issued cookie does not allow users to be logged in
  - Sign-out provides the user the option to delete all tokens that have been issued for this user
  - Cookie expiration time spans, as well as access and refresh token expiration time spans can be adjusted.  
Short time spans provide higher security, longer time spans are more comfortable for users.
  - The message during sign-out has been improved to better communicate to the user what is going to happen
  - SSL usage: redirect to HTTPS is configurable
  - SSL usage: HSTS with configurable max-age header
- External Identity Providers are configurable via the Web UI. Entra ID and Keycloak are integrated so that no plugins must be written.  
Roles of external IDPs are fetched periodically in the background, as well as collected whenever a user authenticates via them.
- Reverse Proxy capabilities have been improved:  
Forwarded headers allow SSL offloading and defining the authority's hostname, so that internal and external applications can use the same instance.
- Available roles/groups are stored in the database, to improve integration of external IDPs.
- A new cmd line option `-ConfigOnly` for configuring the appsettings with an encrypted connection string without touching the database:

```
AuthorizationService.exe setup database -c "connection string..." -p PostgreSQL -
EncryptionCertificateThumbprint MYTHUMBPRINT -ConfigOnly
```

### Breaking Changes

- The nuget package and the namespaces for the TCG Authorization Server interface DLLs has changed.  
If you wrote a plugin, you have to adjust and recompile it.  
This drastic step was taken because we were able to obsolete most reasons for a plugin.
  - The configuration setting to load a plugin changed from `Service->UserStoreAssemblyPath` to `Service->PluginAssemblyPath`
  - The `IClaimsSearcher` interface was dropped.
- Claim changes: a user has the subject claim set to a unique identifier, instead of the username.  
The username value is stored in a 'username' claim.  
The user's display name is stored in the 'name' claim.  
Applications displaying the user's display name or username in a UI must adjust.
- Load balancing requires X-Forwarded-\* headers.  
Please read the documentation section and update the load balancer or reverse proxy configuration.

## 4.1.4 What's new for Version 3.1

- Primus API 3.1
- New component that allows you to achieve single sign-on (SSO) for Primus and its components and external clients.

## 4.2 Copyright

---

Copyright © 2013-2024 TCG Informatik AG, Mühlegasse 18, 6340 Baar, Switzerland.

All rights reserved.

Information in this document is subject to change without notice and does not bear any commitment on the part of TCG Informatik AG. The software described in this document is supplied under a license agreement. The software may only be used or copied in strict accordance with the terms of the agreement. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system or translated into any language in any form by any means without the written permission of TCG Informatik AG. While every precaution has been taken in the preparation of this document, TCG Informatik AG assumes no responsibility for errors, omissions, or for damages resulting from the use of the information herein. Products or corporate names may be trademarks or registered trademarks of other companies and are used only for the explanation and to the owner's benefit, without intent to infringe.