Security Identity Providers Reference

Configure external identity providers (LDAP, OIDC and SAML) for HTML5 and Rich Client operator login.

Security Identity Providers (Reference): configure external authentication mechanisms — LDAP directory servers and OpenID Connect (OIDC) identity providers — that the runtime consults to resolve a user's identity at login time. Each row in this table is a pluggable authentication mechanism.

Identity Providers provide:

- LDAP and Active Directory authentication (RFC 4511) — replaces the legacy single-server `SecurityEnableLDAP` configuration.
- OpenID Connect (OIDC) authentication for HTML5, WPF Rich Client, and Smart Client operator login.
- Per-provider `DefaultPolicy` that maps the authenticated user to one of the solution's Permission Groups.
- Diagnostic counters and last-error tracking, surfaced at runtime via `@Security.IdentityProviders.<Name>`.
- Zero-modification path for PIV and smartcard authentication — configure an OIDC row pointing at a PIV-aware IdP.

Architectural Principle: IdP Identifies; the Platform Manages

An identity provider's only job is to answer **who is logging in**. Once identity is resolved, every downstream concern — session lifetime, Permission Group binding, password and e-sign policies, audit trail, runtime authority — stays with the platform's existing Security machinery.

What an IdP row configures: how to validate credentials against an external system, and which Permission Group the resolved user receives.

What an IdP row does NOT configure:

- **Silent-login or kiosked auto-login** — scripted at the customer's LogOn Display CodeBehind via `@Client.LogOnSSOAsync(providerName)`. The IdP table holds *how to validate credentials*, not *which credentials to silently submit*.
- **Session minting or browser session state** — owned by the platform's HTTP layer. The OIDC callback handler resolves identity, and the platform binds the operator session internally.
- **Multi-factor authentication, conditional access, device posture, anomalous-login detection** — IdP-delivered. The IdP enforces these; the platform receives the resolved identity after the IdP applied them.
- **Audit and compliance posture** — Workspace-scoped. See Security Module Reference for the audit story per Workspace.

The benefit of this layering: every audit, permission, e-sign, session-timeout, and password-policy mechanism continues to work unchanged regardless of which IdP a user signed in through. Customers swap IdPs without touching their Permission Groups, Policies, or Displays.

Configuration Properties

Access at **Security Identity Providers**.

Property	Description	Required	Notes
Name	Unique identifier for the identity provider. Appears on the login screen and in audit references.	Yes	Engineer-defined, for example `CorpAD`, `Keycloak`, `CorpEntraID`.
AuthType	Authentication standard the provider implements. `LDAP` for directory-server auth (RFC 4511); `OIDC` for OpenID Connect token-based auth.	Yes	Dropdown: `LDAP`, `OIDC`, `SAML`. Default `LDAP`.
Active	Provider participates in authentication when true.	No (default `true`)	At most **one** Active row of `AuthType=LDAP` per solution; multiple Active OIDC and SAML rows are allowed.
AuthOptions	Packed `Key=Value;Key=Value` configuration string. AuthType-specific keys (see below).	No	Designer edits this through a type-dispatched dialog — you do not hand-edit the packed string.
Description	Documentation of usage and purpose — which IdP server, which user population, when it was configured.	No	
Category	Classification group for organizing identity providers.	No	

The discriminator column is `AuthType` (specific to this table), not the generic `Type` column reused across many tables — same every-column-unique discipline as `SecuritySecrets.SecretType`.

AuthOptions: Packed Configuration

`AuthOptions` is a `Key=Value;Key=Value` packed string. The Designer's edit dialog presents named fields per AuthType and writes the packed form on save; you only handle the packed string directly when scripting or inspecting.

LDAP keys

Key	Description	Required

`Server`	The directory server. From FrameworX 10.1.6 any of these forms works: a host name (`ad.corp.local`), a host and port (`ad.corp.local:389`), or an address (`ldap://ad.corp.local:389`, `ldaps://ad.corp.local:636`). In 10.1.5 and earlier only the host-name forms reached the directory; a server entered as an `ldap://` address was never contacted. For `ldaps://`, the computer running the runtime must trust the directory's certificate, and the name you enter must match the name in that certificate (an IP address usually does not).	Yes
`DefaultPolicy`	Name of the SecurityPermissions group applied to authenticated users when no group-name match is found. Empty falls through to the platform's default behavior.	No

Example LDAP `AuthOptions`:

```
Server=ldaps://ad.corp.local:636;DefaultPolicy=Operators
```

OIDC keys

Key	Description	Required
`Authority`	IdP base URL. **FrameworX appends `/.well-known/openid-configuration` automatically — if your IdP gives you the full discovery (metadata) URL, remove that suffix.** Examples: `https://login.microsoftonline.com/<tenant-id>/v2.0` (Entra ID), `https://accounts.google.com` (Google), `https://keycloak.corp.local/realms/factory` (self-hosted Keycloak).	Yes
`ClientId`	OAuth 2.0 client_id registered at the IdP for this FrameworX deployment.	Yes
`ClientSecretRef`	`/secret:<Name>` reference to a Security Secrets row holding the OAuth client_secret. Resolved server-side via the Secrets resolver — the plaintext never appears in `AuthOptions`.	Yes for confidential clients
`RedirectUri`	OAuth redirect URI registered at the IdP. Must end in `/oidc/callback`. Example: `https://factory.corp.local/oidc/callback`. On the HTML5 Web Client it must also share the **origin** (scheme, host, port) with the URL the client is served from — see *Authentication Flows* below.	Yes
`Scopes`	Comma-separated OAuth scopes. Standard set: `openid,profile,email,groups`.	Yes
`UsernameClaim`	Name of the OIDC claim whose value becomes the FrameworX username. Common: `preferred_username`, `email`, `upn`.	Yes
`GroupsClaim`	Name of the OIDC claim whose value list maps to Permission Group names. Common: `groups`, `roles`.	No
`DefaultPolicy`	Permission Group applied when no group claim matches an existing SecurityPermissions name. Empty falls through to platform default.	No
`SSL`	Requires HTTPS for all communication with the IdP — discovery document, JWKS, and the authorization-code / token exchange. `true` is the default and the correct production setting. Set it to `false` only to test or develop against an identity provider published over plain HTTP (for example a local Keycloak on `http://localhost:8080`); with `SSL=false` the tokens and the client secret travel unprotected, so it must never be used in production.	No (default `true`)

In the Designer, `SSL` is the **SSL** checkbox of the Identity Provider edit dialog — checked by default. A row saved before this option existed carries no `SSL` key and is treated as `true`, so upgraded solutions stay HTTPS-mandatory with no edit.

Example OIDC `AuthOptions`:

```
Authority=https://keycloak.corp.local/realms/factory;ClientId=fwx-runtime;ClientSecretRef=/secret:
KeycloakClientSecret;RedirectUri=https://factory.corp.local/oidc/callback;Scopes=openid,profile,email,groups;
UsernameClaim=preferred_username;GroupsClaim=groups;DefaultPolicy=Operators;SSL=true
```

SAML keys

From FrameworX 10.1.6, a SAML row uses the keys `DefaultPolicy`, `IdpEntityId`, `IdpSsoUrl`, `IdpCertificate`, `SpEntityId`, `AcsUrl`, `UsernameClaim`, `GroupsClaim` and `SSL`. The identity provider must sign the assertion with SHA-256 or stronger. Encrypted assertions, single logout and sign-in started from the identity provider's portal are not supported.

Authentication Flows

A single `AuthType=OIDC` row serves every client path — the configuration is identical for all of them; the platform handles the differences transparently.

OIDC sign-in has **one entry point** on every client: `@Client.LogOnSSOAsync(providerName)`, called from the LogOn Display CodeBehind. The platform owns the whole round-trip — opening the IdP, validating the token, resolving the identity and binding the runtime session. No token handling is written in the solution.

```
// LogOn Display CodeBehind - same call on HTML5, Rich Client and Smart Client
int rc = await @Client.LogOnSSOAsync("CorpEntraID");
if (rc != 0)
    @Client.UI.ShowMessageBox("SSO sign-in failed: " + rc);
```

The `providerName` argument is the `Name` of the Identity Provider row. On Rich Client and Smart Client it may be omitted when the solution has exactly one Active OIDC row, in which case the platform picks it; on the HTML5 Web Client the name is always required.

HTML5 (Web browser)

1. Operator opens the LogOn Display in a browser.
2. Operator clicks the **Sign in with SSO** button and picks an Identity Provider from the dropdown.
3. The Display CodeBehind calls `@Client.LogOnSSOAsync("<Name>")`. The platform opens the IdP in a popup window and keeps the client tab — and everything the operator had open in it — alive.
4. Operator authenticates in the popup (password, MFA, PIV, WebAuthn — whatever the IdP enforces).
5. The IdP returns to `/oidc/callback`. The platform validates the token, resolves identity, closes the popup, and binds the runtime session.
6. `@Security.CurrentUser` updates and the normal operator displays open.

Popup blocked. If the browser refuses the popup, the platform falls back to running the flow in the current tab. The sign-in still completes, but the client reloads on return, so the operator lands back on the startup Display instead of the one they were on. Allowing popups for the runtime URL avoids this.

Deployment requirement. `RedirectUri` must share its **origin** — scheme, host *and* port — with the URL the HTML5 client is served from. `http://localhost:3201` and `http://127.0.0.1:3201` are different origins: with that mismatch the operator authenticates successfully at the IdP and still ends up not signed in.

WPF Rich Client and Smart Client (OIDC Native-Flow SSO)

The same call, adapted per RFC 8252 (OAuth 2.0 for Native Apps): the client opens the operator's **system browser** (not an embedded WebView — enterprise IdPs reject those) and listens on a **loopback HTTP endpoint** for the redirect.

1. Operator clicks **Sign in with SSO** on the Rich Client LogOn Display, and the CodeBehind calls `@Client.LogOnSSOAsync("<Name>")`.
2. The client opens the system default browser at the IdP, with the redirect pointed at its own loopback endpoint (PKCE-protected).
3. Operator authenticates against the IdP in the system browser.
4. The IdP redirect lands on the loopback endpoint the client is listening on; the platform validates the token, resolves identity, and binds the session.
5. The call returns `0` and `@Security.CurrentUser` reflects the signed-in operator.

Native flow is available on Rich Client and Smart Client (Windows + WPF). It is not available, and not needed, on the HTML5 Web Client, which already has a browser.

LDAP

LDAP rows have no callback flow. The user types username and password into the LogOn Display; the platform forwards the credentials to the configured LDAP `Server`; on success the resolved user receives the row's `DefaultPolicy` (or a matched Permission Group).

- **User name forms.** A plain name is looked up as the directory account name. A name with `@` (`user@corp.local`) is looked up as that account's user principal name; the text before `@` alone never selects another account (from 10.1.6). The user name is escaped in the directory search, so characters such as `( ) * \` are compared, not interpreted.
- **Directory unreachable.** From 10.1.6 the sign-in returns an identity-provider error (result code 17) instead of "invalid user", and the provider's `LastErrorMessage` shows the cause (see Runtime Diagnostic Properties). A wrong password still returns "invalid user name or password".

Configuration Examples

Active Directory via LDAP

Field	Value
Name	`CorpAD`
AuthType	`LDAP`
Active	`true`
AuthOptions	`Server=ldaps://ad.corp.local:636;DefaultPolicy=Operators`
Description	Corporate Active Directory for factory operators

Microsoft Entra ID (Azure AD) via OIDC

1. Register an application in Entra ID; note the **Application (client) ID** and **Directory (tenant) ID**.

2. Add a **Web** redirect URI: `https://factory.corp.local/oidc/callback`.
3. Generate a client secret; store it in Security Secrets as `EntraIDClientSecret`.
4. In Security Identity Providers, create a row:

Field	Value
Name	`CorpEntraID`
AuthType	`OIDC`
Active	`true`
AuthOptions	`Authority=https://login.microsoftonline.com/<tenant-id>/v2.0;ClientId=<application-client-id>;ClientSecretRef=/secret:EntraIDClientSecret;RedirectUri=https://factory.corp.local/oidc/callback;Scopes=openid,profile,email,groups;UsernameClaim=preferred_username;GroupsClaim=groups;DefaultPolicy=Operators;SSL=true`

Keycloak (self-hosted) via OIDC

1. In Keycloak, create a realm (e.g., `factory`).
2. Create a client `fwx-runtime` with **client authentication** on, redirect URI `https://factory.corp.local/oidc/callback`.
3. Copy the generated client secret; store it in Security Secrets as `KeycloakClientSecret`.
4. In Security Identity Providers, create a row pointing at the Keycloak realm:

Field	Value
Name	`Keycloak`
AuthType	`OIDC`
Active	`true`
AuthOptions	`Authority=https://keycloak.corp.local/realms/factory;ClientId=fwx-runtime;ClientSecretRef=/secret:KeycloakClientSecret;RedirectUri=https://factory.corp.local/oidc/callback;Scopes=openid,profile,email,groups;UsernameClaim=preferred_username;GroupsClaim=groups;DefaultPolicy=Operators;SSL=true`

A Keycloak instance running on plain HTTP for development (`http://localhost:8080/realms/factory`) additionally needs `SSL=false` — uncheck **SSL** in the edit dialog. Re-check it before the solution reaches production.

Auth0 via OIDC

1. In the Auth0 dashboard, create a **Regular Web Application**; note the **Domain**, **Client ID**, and **Client Secret**.
2. Add an **Allowed Callback URL**: `https://factory.corp.local/oidc/callback`.
3. Store the client secret in Security Secrets as `Auth0ClientSecret`.
4. If group-to-Permission Group mapping is needed, add a **roles** claim to the ID token via an Auth0 Action (Auth0 does not emit a `groups` claim by default).
5. In Security Identity Providers, create a row:

Field	Value
Name	`Auth0`
AuthType	`OIDC`
Active	`true`
AuthOptions	`Authority=https://<tenant>.auth0.com/;ClientId=<application-client-id>;ClientSecretRef=/secret:Auth0ClientSecret;RedirectUri=https://factory.corp.local/oidc/callback;Scopes=openid,profile,email;UsernameClaim=email;GroupsClaim=roles;DefaultPolicy=Operators;SSL=true`

login.gov via OIDC

login.gov is the U.S. federal single sign-on service. Two operational specifics distinguish it from a typical OIDC IdP: client authentication is by signed JWT assertion (`private_key_jwt`), not a shared client secret, and the IdP does not emit a groups claim — every authenticated user receives the row's `DefaultPolicy`.

1. Register the FrameworX deployment in the login.gov partner dashboard (sandbox: `https://idp.int.identitysandbox.gov`; production: `https://secure.login.gov`). Upload the public key of the JWT signing certificate FrameworX will use for client assertion.
2. Note the assigned **Client ID** and select the IAL / AAL level appropriate to the deployment (IAL1 / AAL1 for basic identity; IAL2 / AAL2 when verified identity and MFA are required).

3. Add the redirect URI `https://factory.corp.local/oidc/callback` in the partner dashboard.
4. Store the JWT signing key reference in Security Secrets as `LoginGovSigningKey` — resolved server-side at client-assertion time.
5. In Security Identity Providers, create a row:

Field	Value
Name	`LoginGov`
AuthType	`OIDC`
Active	`true`
AuthOptions	`Authority=https://secure.login.gov;ClientId=<client-id>;ClientSecretRef=/secret:LoginGovSigningKey;RedirectUri=https://factory.corp.local/oidc/callback;Scopes=openid,email;UsernameClaim=email;DefaultPolicy=Operators;SSL=true`

login.gov does not pass group claims, so omit `GroupsClaim` — every authenticated user falls through to the row's `DefaultPolicy` (typically `Operators`). Use the sandbox Authority (`https://idp.int.identitysandbox.gov`) for development; switch to `https://secure.login.gov` for production.

PIV and smartcard authentication

PIV is delivered through an OIDC IdP that supports certificate-based authentication (Entra ID with CBA, AD FS with smartcard, login.gov, or any OIDC-compliant IdP fronting a PIV PKI). Configure an `AuthType=OIDC` row pointing at the PIV-aware IdP — no special FrameworX configuration is needed. The IdP enforces the PIV cert; FrameworX receives the resolved identity.

Group Claim to Permission Group Mapping

Users defined in the solution. From FrameworX 10.1.6, when the signed-in user name matches a user in the solution, either a Security User defined in Designer or a user created at runtime, that user's own Permission Groups apply and the group claims sent by the provider are ignored. The user gets the same permissions as with a password login, and no per-provider role mapping is needed. This applies to the HTML5 client and the WPF Rich Client. The group-claim mapping below applies to users who are not in the solution.

When the OIDC IdP includes a groups claim in the ID token (per the `GroupsClaim` configuration), FrameworX iterates each group name in order and matches it case-exactly against the names of `SecurityPermissions` groups in the solution.

The first match wins. If no group name matches, the row's `DefaultPolicy` applies. If `DefaultPolicy` is empty, the platform's default behavior applies.

Practical mapping pattern: name your Permission Groups in FrameworX to match the IdP's group names exactly. Customers commonly create groups in their IdP named `Operators`, `Engineers`, `Supervisors` to match Permission Groups of the same names; then no per-provider mapping configuration is needed.

LDAP-side group mapping follows the same rule against the LDAP user's group memberships. From 10.1.6 the groups read are always those of the account that signed in.

LDAP group mapping — requirements and limits (10.1.6)

- **Run the runtime under a domain identity.** The runtime reads the directory groups partly with its own Windows identity. Run it as a service (Local System) on a computer joined to the domain, or under a domain account. Under a *local* Windows account the sign-in still succeeds, but the directory groups may not be read, and the user then gets only the `DefaultPolicy` / Guest permissions.
- **.NET 10 (Multiplatform) runtime: the primary group is not returned.** On the .NET 10 runtime the user's primary group (normally `Domain Users`) is not part of the groups read, so a Permission Group named `Domain_Users` never matches there; the .NET Framework 4.8 (Windows) runtime does return it. Map a regular directory group instead (for example `FX_Operators`), which works on both runtimes.

Group names with spaces. Identity providers often use spaces in group names, for example `Domain Admins`. From FrameworX 10.1.6, Permission Group names cannot contain spaces, and a group claim with spaces also matches the Permission Group named with underscores in place of the spaces, `Domain_Admins`. The claim exactly as sent is tried first, so groups whose names already match keep resolving as before.

Runtime Diagnostic Properties

Each Identity Provider exposes runtime metrics readable from scripts and Display CodeBehind via `@Security.IdentityProviders.<Name>`:

Property	Type	Meaning
`LastAuthSuccessTime`	DateTime	Timestamp of the most recent successful auth via this provider.
`LastAuthErrorTime`	DateTime	Timestamp of the most recent auth failure.
`LastErrorMessage`	String	Diagnostic message from the most recent failure (IdP error text, certificate failure, JWKS fetch error, etc.).
`TokensValidatedCount`	Integer	Count of successful token validations since module start.

`TokensFailedCount`	Integer	Count of failed token validations since module start.
`DiscoveryDocStatus`	String	OIDC only: `OK` or last fetch error.
`JwksKeysCount`	Integer	OIDC only: number of signing keys loaded from the IdP's JWKS endpoint.
`IsHealthy`	Boolean	Composite indicator: `true` when discovery is `OK` and recent auths are succeeding.

These properties are runtime-only — they reset to zero on module restart. They do not persist across solution reload. Use them to drive an operator status panel showing IdP health.

Migration from Pre-10.1.5

Solutions upgraded from pre-10.1.5 automatically migrate the legacy LDAP configuration:

- The `SecurityEnableLDAP` and `SecurityLDAPServer` columns on the pre-10.1.5 `SecurityRuntimeSettings` table are read, then materialized as a single `AuthType=LDAP, Active=true` row in `SecurityIdentityProviders` with `AuthOptions = "Server=<old-server>;DefaultPolicy=AD;"`.
- The legacy columns are then dropped from `SecurityRuntimeSettings`.
- Migration runs once on first open of the solution under 10.1.5+.

No script or Display change is needed to keep LDAP login working post-upgrade — the runtime LDAP sign-in now reads the new table and produces identical behavior. Customers wanting to add OIDC providers add rows alongside the migrated LDAP row.

Common Mistakes

Mistake	Why it fails	Fix
Adding a `Default=true` or silent-login flag on a row	The IdP table holds *how to validate credentials*, not *which credentials to silently submit*.	Script the sign-in at the LogOn Display CodeBehind via `@Client.LogOnSSOAsync(providerName)`.
Signing in through `@Security.LogOn` / `LogOnAsync` for an OIDC provider	Those entry points take a username and a password and encrypt the password before transport — there is no password in a federated sign-in.	Use `@Client.LogOnSSOAsync(providerName)`, the dedicated federated entry point. It performs the whole exchange and binds the session.
`RedirectUri` on a different origin than the HTML5 client URL	The sign-in window and the client tab end up on different origins and cannot exchange the result — the operator authenticates at the IdP and is still not signed in.	Use the same scheme, host and port in `RedirectUri` as in the URL operators open the client with (`localhost` and `127.0.0.1` are not interchangeable).
`Authority` includes `/.well-known/openid-configuration`	FrameworX appends the suffix again, and discovery fails with `IDX20807` on a URL with the suffix duplicated.	Remove the suffix. For Entra ID use `https://login.microsoftonline.com/<tenant-id>/v2.0`.
A reverse proxy in front of the runtime rewrites redirect headers (for example IIS ARR with *Reverse rewrite host in response headers* enabled)	The proxy replaces the IdP host in the sign-in redirect with its own host. The browser then downloads an empty file named `authorize` instead of opening the IdP sign-in page.	In IIS: Application Request Routing Cache Server Proxy Settings uncheck *Reverse rewrite host in response headers*, or restrict any `Location` rewrite rule to the internal runtime host. The proxy must also forward `/oidc/start` and `/oidc/callback` to the runtime.
Entra ID shows *Need admin approval*	The tenant does not allow users to consent to the app's permissions.	An Entra administrator grants admin consent: App registrations *API permissions* *Grant admin consent*.
Leaving SSL unchecked in production	Discovery and the token exchange then run over plain HTTP: ID tokens and the client secret cross the network unprotected and can be intercepted or replaced.	Keep **SSL** checked (the default). Uncheck it only against a development IdP published over HTTP.
Multiple `Active=true` LDAP rows	At most one Active LDAP row per solution (preserves the 10.1.4 single-LDAP-server semantic). Designer save validator rejects; runtime takes first-by-ID-order on duplicates and logs a warning.	Mark all but one LDAP row `Active=false`, or split into separate solutions.
Plain-text password in `ClientSecretRef`	The OIDC validator resolves `ClientSecretRef` through the Security Secrets resolver. A plain value or unrecognized prefix is rejected.	Create a Security Secrets row holding the client secret; reference it as `/secret:<Name>`.
LDAP: the runtime runs under a local Windows account	The directory groups are read partly with the runtime's own identity; a local account usually cannot query the directory, so users sign in with only the default / Guest permissions.	Run the runtime as a service (Local System) on a domain-joined computer, or under a domain account.
LDAP: a Permission Group named `Domain_Users` on the .NET 10 runtime	The .NET 10 runtime does not return the primary group (`Domain Users`).	Map a regular directory group instead; it works on both runtimes.
LDAP: `ldaps://` with an IP address or an untrusted certificate	The TLS connection fails and the sign-in returns an identity-provider error.	Use the directory's certificate name in `Server` and make the runtime computer trust the issuing CA.

Group names in IdP do not match Permission Group names	Group-to-Permission matching is case-exact on the name.	Either rename Permission Groups to match IdP group names, or rename IdP groups to match FrameworX Permission Group names. From FrameworX 10.1.6, for an IdP group name with spaces, name the Permission Group with underscores in place of the spaces.

See Also

- Security Module Reference — complete security documentation.
- Security Secrets Reference — used by `ClientSecretRef` for OIDC client secrets.
- Security Permissions Reference — the Permission Groups that group claims and `DefaultPolicy` map to.
- Security Policies Reference — session timeout, password rules, e-sign — applied post-identity-resolution regardless of IdP.

In this section...