Security Identity Providers Reference

Configure external identity providers (LDAP and OIDC) 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.IdentityProvider.<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 `@Security.LogOnWithTokenAsync (providerName, idToken)`. 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 hands off via a one-time GUID; the standard `/start` flow mints the operator session.
- **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`. 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 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`	LDAP server URL (e.g., `ldap://ad.corp.local:389` or `ldaps://ad.corp.local:636`).	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 discovery URL — the base of `/.well-known/openid-configuration`. 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`.	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

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
```

Authentication Flows

A single `AuthType=OIDC` row serves three client paths — the configuration is identical for all three; the platform handles the differences transparently.

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 navigates the browser to `/oidc/start?provider=<Name>&returnUrl=/`.
4. The platform redirects to the IdP. Operator authenticates (password, MFA, PIV, WebAuthn — whatever the IdP enforces).
5. IdP redirects back to `/oidc/callback`. The platform validates the token, resolves identity, and 302-redirects the browser to the LogOn Display with a one-time `identityToken` GUID.
6. The Display CodeBehind reads `@Client.QueryParams["identityToken"]` and calls `@Security.LogOnWithTokenAsync (providerName, identityToken)`.
7. Session is bound, `@Security.CurrentUser` updates, normal operator displays open.

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

The same flow, 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.
2. Rich Client opens the system default browser to `/oidc/start?provider=<Name>&returnUrl=http://127.0.0.1:<port>/handoff`.
3. Operator authenticates against the IdP in the system browser.
4. The IdP redirect lands on the loopback `/handoff` endpoint that the Rich Client is listening on.
5. Rich Client extracts the `identityToken` GUID, hands it to the orchestrator, and the session is bound — same `@Security.LogOnWithTokenAsync` path as the browser flow.

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).

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`

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`

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`

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`

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

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.

Runtime Diagnostic Properties

Each Identity Provider exposes runtime metrics readable from scripts and Display CodeBehind via `@Security.IdentityProvider.<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 `CheckUser` LDAP branch 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 silent-login at the LogOn Display CodeBehind via `@Security.LogOnWithTokenAsync (providerName, idToken)`.
Routing tokens through `@Security.LogOn` / `LogOnAsync`	Those entry points encrypt the incoming password before transport, which mangles JWTs.	Use the parallel `@Security.LogOnWithTokenAsync (providerName, idToken)` entry point.
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>`.
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.

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...