Security Secrets Reference

Securely store credentials and secrets.

Security Secrets (Reference): Securely store and manage sensitive credentials such as passwords, API keys, and connection strings without exposing plain-text values in your configuration.

Security Secrets provide:

- Centralized credential storage
- Encrypted storage for Password and BearerToken values, decrypted server-side only
- Integration with Datasets (DB connections)
- Integration with Devices/TagProviders (MQTT, OPC UA, etc.)
- Administrative control over sensitive values

Configuration Properties
Secret Reference Syntax
Usage in Configurations
Configuration Examples
Security Considerations
Best Practices
Migration and Portability
Troubleshooting
See Also

Configuration Properties

Access at: **Security** **Secrets**

Property	Description	Required	Access Control
Name	Unique identifier for the secret. Referenced from other tables as `/secret:<Name>`.	Yes	All users
SecretType	Type of credential (Password, APIKey, Token). All three types resolve through the same mechanism. The type sets how the value is stored.	Yes	All users
SecretValue	Credential value. Password and BearerToken values are stored encrypted. APIKey and Token values are stored as plain text.	Yes	**Administrators only**
Description	Documentation of usage/purpose	No	All users
BearerToken	When set, the resolved secret is formatted for use as an HTTP `Authorization: Bearer <value>` header. Use for REST APIs, OAuth-protected endpoints, and OIDC client tokens that require bearer-token presentation rather than basic-auth credentials.	No	**Administrators only**
Associated User	Binds the secret to a specific runtime user (by Name from `SecurityUsers`). When set, only the named user's session can resolve the secret — access from other sessions returns the unresolved reference. Use to scope an API key, personal access token, or per-operator credential to one identity.	No	**Administrators only**
Enabled	Boolean flag. When `false`, the secret is retained but every `/secret:<Name>` reference behaves as if missing — the resolver returns the raw token and the downstream connection fails. Use to rotate, audit, or temporarily revoke a credential without deleting the row and losing its Description and historical context.	No (default `true`)	**Administrators only**

Important: Only administrators can view or edit SecretValue, BearerToken, AssociatedUser, or Enabled. Standard users can reference secrets in configurations but cannot see the actual values.

UI Behavior

- **Cell masking by default.** The `SecretValue` and `BearerToken` cells render masked (e.g., `********`) in the Designer grid for every user, including administrators. The plaintext is only revealed when an administrator explicitly chooses to reveal it through the cell context menu.
- **Admin-only Copy-to-clipboard context menu.** Right-click on a masked secret cell opens a context menu with a **Copy to clipboard** entry, available only to users with administrator privileges. The action copies the decrypted plaintext to the OS clipboard for one-shot use (for example, pasting into an external test tool). The clipboard write is recorded in the AuditTrail.

Secret Reference Syntax

A secret is **referenced** (never pasted inline) from any password or credential field using a single, platform-wide syntax:

```
/secret:<Name>
```

The colon form is the stored syntax that FrameworX resolves at runtime. Older documentation showing a dash form (`/secret-<Name>`) is stale — use the colon form.

In the Designer, most fields that accept a secret reference render with a dropdown picker that writes the correct `/secret:<Name>` value for you; the field may also display a friendly label like `[Secret: <Name>]` — that is the UI label, while the stored value is always `/secret:<Name>`.

Embedded form (composite strings)

Inside a larger configuration string (such as a connection string that mixes plain text and one or more secrets), the embedded form is used:

```
#</secret:<Name>/secret>
```

The same server-side resolver decrypts both forms.

Where it resolves

Resolution happens server-side during value decryption. Password-bearing fields that already flow through the platform's decryption pipeline get secret resolution automatically — consumer modules (Dataset, Device, TagProvider, Historian) do not implement their own lookup. A missing or misnamed reference silently returns the raw text unchanged, which means the downstream connection attempt will fail as if the password were wrong; check the audit log to diagnose.

Usage in Configurations

Datasets Integration

When configuring database connections:

1. Navigate to **Datasets DBs**
2. In the **LogonPassword** field, either type the reference directly or click the secret picker
3. The field stores `/secret:<Name>`; the Designer may display it as `[Secret: <Name>]`
4. Connection uses the decrypted value at runtime

Stored value example:

```
Database: RuntimeUsers
Authentication: SQL Server Auth
Username: dbuser
LogonPassword: /secret:DBRuntimePass
```

See Datasets DBs Reference for the complete DB-row schema.

Device/TagProvider Integration

For MQTT, OPC UA, and other protocols:

1. Navigate to **Devices Channels** or **TagProviders**
2. In authentication fields, use the secret reference syntax
3. Format: `/secret:<SecretName>`

Example:

```
MQTT Broker Authentication:
Username: mqttuser
Password: /secret:MQTTBrokerPass
```

Script Access

> For security reasons, there is no access to secrets via script, only through the configuration UI in each scenario where secrets can be applied.

Configuration Examples

Basic Secret Setup

Name	SecretType	SecretValue	Description

DBProductionPass	Password	********	Production SQL Server
MQTTBrokerPass	Password	********	MQTT cloud broker

Runtime Users Database Example

1. Create secret:
   - Name: `RuntimeUsersDBPass`
   - Type: `Password`
   - Value: `[encrypted]`
   - Description: `RuntimeUsers SQL authentication`
2. Use in Dataset:
   - Go to **Datasets DBs RuntimeUsers**
   - Set the **LogonPassword** field to: `/secret:RuntimeUsersDBPass`

Security Considerations

Access Control Matrix

Action	Administrator	Engineer	Guest
Create Secret	Yes	Yes	Yes
Edit SecretValue	Yes	No	No
View SecretValue	No	No	No
Reference in Config	Yes	Yes	Yes
View Secret List	Yes	Yes	Yes

Encryption Details

- Password and BearerToken values are stored encrypted in the solution file
- APIKey and Token values are stored as plain text in the solution file
- Decrypted only at runtime, server-side
- Never logged in plain text
- Not included in solution exports — the target system must recreate values

---

Best Practices

Naming Conventions

Pattern	Example	Use For
`DB<System>Pass`	`DBHistorianPass`	Database passwords

Implementation Guidelines

1. **Create secrets before configuration**
   - Define all secrets in the Security module first
   - Then reference in other modules using `/secret:<Name>`
2. **Use descriptive names**
   - Good: `ProductionSQLPass`
   - Bad: `Secret1`
3. **Document usage**
   - Always fill the Description field
   - Include where the secret is used
   - Note rotation schedule if applicable
4. **Regular rotation**
   - Change production secrets periodically
   - Update the SecretValue — references stay the same
   - Test thoroughly after changes
5. **Environment separation**
   - Use different secrets for Dev/Test/Production
   - Name accordingly: `DevDBPass`, `ProdDBPass`

Migration and Portability

Exporting Solutions

When exporting solutions:

- Secret names and types are exported
- Secret values are NOT exported
- The target system must recreate values
- `/secret:<Name>` references in consumer fields carry over unchanged — recreate the secrets with the same Names to keep references valid

Import Process

1. Import solution to new system
2. Navigate to **Security Secrets**
3. For each secret, enter appropriate value
4. Test all connections

Troubleshooting

Common Issues

Issue	Cause	Solution
Connection fails as if password is wrong	Reference uses the older dash form `/secret-<Name>` (not recognized) or the Name is misspelled	Use the colon form `/secret:<Name>`; verify the exact Secret Name
"Access denied" when editing a secret	Insufficient permissions	Use administrator account
Connection fails after rotation	New SecretValue incorrect or caching	Verify and re-enter value; restart the consuming connector or reload the project
Secret list empty	Filter applied	Check module filter settings

Validation

Test secret configuration:

1. Create test secret
2. Reference in test connection as `/secret:<Name>`
3. Verify connection succeeds
4. Check runtime logs for errors

Audit Trail

Secret operations are logged:

- Creation/modification timestamp
- User who made changes
- NOT the actual values

See Also

- Security Module Reference — Complete security documentation
- Datasets DBs Reference — LogonPassword accepts `/secret:<Name>`
- Security Module How-to — Security guidelines

In this section...
