Security Users Reference

Create and manage platform users.

Reference **Modules** **Security** **UI** RuntimeUsers | Users | Permissions | Policies | Identity Providers | Secrets | Monitor

---

Security Users (Reference) manages user accounts, authentication, and access control throughout the solution. A SecurityUser defines:

- Named user accounts with credentials
- Permission group assignments
- Security policy enforcement
- Access control levels
- User lifecycle management

The system includes pre-defined users and supports enterprise authentication methods.

Pre-Defined Users
Configuration Properties
Guest Access
Administrator Privileges
User Lifecycle Management
Runtime Authentication
Permission Integration
Security Policies
User Properties Access
Best Practices
Troubleshooting

Pre-Defined Users

Four system users are configured by default:

User	Purpose	Default Password	Notes
Administrator	System control and security management	None	Set password immediately
Guest	Anonymous access and default logout	None	Cannot add password
User	Generic authenticated access	None	Template for new users
MCP	AI agent identity for DesignerMCP / ConsoleMCP / RuntimeMCP / RuntimeMCPHttp connectors	None	Predefined hardcoded user, parallel to Guest and Administrator. The connector authenticates as this user via `/UserName` + `/Password` CLI args. Cannot be deleted. Permission Group governs what the AI can do; `UserComputer` column on TrackChanges disambiguates multi-machine MCP sessions.

Do not delete or modify the row IDs of these built-in users. Do not create duplicate users with these names.

Configuration Properties

Property	Description	Required
Name	Unique username for login	Yes
Permissions	Permission groups (comma-separated)	Yes
Password	Encrypted user password	No
PasswordHint	Password recovery hint	No
Policy	Security policy assignment	No
Deleted	Soft delete flag	No
Alias	Alternative identifier	No
Company	Organization association	No
UserGroup	Department/group assignment	No
Attributes	Custom user properties	No
Level	Hierarchical access level (0-255)	No
Category	User classification	No
LockState	Account lock status	Auto
ContactInfo	Email, phone, details	No

Guest Access

The Guest user provides anonymous access:

- Active when no user logged in
- Default after logout
- No password capability
- Cannot be deleted
- Permissions define anonymous access level

Configure Guest permissions carefully to secure anonymous access.

Administrator Privileges

Exclusive Administrator capabilities:

- Delete users permanently
- Block/unblock accounts
- Set database passwords
- Delete audit trails
- Modify security policies
- Override permission inheritance

User Lifecycle Management

Creating Users

1. Navigate to **Security Users**
2. Click first row to add
3. Required fields:
    - **Name**: Unique identifier
    - **Permissions**: At least one group
4. Optional security:
    - **Password**: Meet policy requirements
    - **Policy**: Assign security level

Disabling Users

Three methods for removing access:

Method	Effect	Use Case	Reversible
Block	Prevents login	Temporary suspension	Yes
Flag Deleted	Blocks + marks deleted	Audit trail preservation	Yes
Delete	Permanent removal	Complete cleanup	No

Password Management

Runtime password changes always require the user's current password. There is no script API that sets a password directly — the user's `Password` field is not writable from script. To issue credentials administratively, create the user with a one-time password (below) or use the runtime user-management UI.

> Only **runtime users** can have their password changed at runtime. Users configured in the Designer under **Security Users** are read-only at runtime; attempting to change one returns status **2**.

Change a password:

```
// Requires the current password. Returns true on success.
bool ok = await @Client.ChangeUserPasswordAsync("username", "oldPassword", "newPassword");
```

Change a password with a detailed status code:

```
TRef<int> status = new TRef<int>();
bool ok = await @Client.ChangeUserPasswordWithStatusAsync("username", "oldPassword", "newPassword", status);
if (!ok)
    @Info.Trace("Password change failed. Status: " + status.Value);
```

Status	Meaning
0	Password updated successfully
1	New password must differ from the current password
2	Designer-configured users cannot change password at runtime
3	Runtime user not found
4	Invalid current password
5	The user's policy has `AllowPasswordChange` disabled
6	New password shorter than the policy's `PasswordMinLength`
7	Blocked by the policy's `MinPasswordAge`
8	Blocked by the policy's `PasswordHistory` — password already used
9	Server not connected
10	Error writing the new password to the database

Check whether the signed-in user must change their password:

```
// Read-only flag. Raised by a one-time password, or by the policy's MaxPasswordAge.
if (@Client.CurrentUser.ChangePasswordRequired)
{
    // Open your change-password display
}
```

Create a user who must change the password on first login:

```
// The final argument (oneTimePassword) raises ChangePasswordRequired on the new user.
// Returns an empty string on success, or the error message on failure.
string error = await @Security.AddRuntimeUserAsync(
    "username", "Operator", "initialPassword", "", "Enhanced", "", "", "", true);
if (error.Length > 0)
    @Info.Trace("Could not create user: " + error);
```

Retrieve a password hint:

```
string hint = await @Security.GetPasswordHintAsync("username");
```

When `ChangePasswordRequired` is set, the policy's `AllowPasswordChange` and `MinPasswordAge` restrictions are bypassed so the user can always complete the mandatory change.

Runtime Authentication

Login Methods

Form-Based Login:

```
// LogOnAsync returns 0 on success, or a non-zero eSecurityErrors code on failure.
int rc = await @Client.LogOnAsync("username", "password");
if (rc == 0)
    @Info.Trace("User logged in: " + @Client.UserName);
else
    @Info.Trace("Logon failed. Code: " + rc);
```

Windows Authentication:

```
// Windows Authentication is enabled in the solution's Security settings, not from script.
// When AD-mapped authentication is active, the signed-in Windows identity is exposed as a
// SecurityUser via @Security.WindowsUser (null on view-clients without Windows authentication).
if (@Security.WindowsUser != null)
    @Info.Trace("Windows user: " + @Security.WindowsUser.Name);
```

External Authentication (Identity Providers):

```
// External auth uses the Identity Providers table (OIDC / OAuth2 / SSO), configured under
// Security -> Identity Providers. Each provider row exposes AuthType, Server, Authority,
// ClientId, ClientSecretRef, RedirectUri, Scopes, UsernameClaim, GroupsClaim.

// Federated sign-in from a Display CodeBehind (provider name from that table):
int rc = await @Client.LogOnSSOAsync("CorpAD");
if (rc != 0)
    @Info.Trace("SSO logon failed. Code: " + rc);
```

Permission Integration

Users inherit permissions from assigned groups:

```
User: John
Permissions: Operator, Maintenance
Result: Combined permissions from both groups
```

See [Security Permissions] for group configuration.

Security Policies

Policies enforce password and session rules:

```
User: Mary
Policy: Enhanced
Result: Strong password, 90-day expiration, session timeout
```

See [Security Policies] for policy configuration.

User Properties Access

Runtime Properties

```
// Signed-in user and session data
string currentUser = @Client.UserName;
string clientIP    = @Client.Session.ComputerIP;

// Permission / level info lives on the user object (@Security.User is the collection):
var user           = @Security.User[@Client.UserName];
string permissions = user.PermissionsName;   // e.g. "Operator;Maintenance;"
string level       = user.Level;             // Level is a string

// Authorization checks are bitmask / level based (no arbitrary permission-name string):
bool okLevel = user.CheckLevel(50);
```

User Management

```
// Read a user (@Security.User is the runtime user collection)
var user = @Security.User["username"];
string company = user.Company;      // writable
string group   = user.UserGroup;    // writable

// Writable profile fields: Alias, Company, UserGroup
user.Company   = "ACME";
user.UserGroup = "Maintenance";
user.Alias     = "jdoe";

// NOTE: ContactInfo and Level are read-only at runtime (internal setters); Level is a string.
```

Best Practices

1. **Set Administrator password** - Immediately on deployment
2. **Use permission groups** - Don't assign individual permissions
3. **Apply security policies** - Enforce password standards
4. **Audit user changes** - Track modifications
5. **Review Guest permissions** - Minimize anonymous access
6. **Document user roles** - Clear responsibility matrix
7. **Regular cleanup** - Remove inactive users

Troubleshooting

Cannot login:

- Verify username/password
- Check account not blocked
- Confirm permissions assigned
- Review policy restrictions

Password issues:

- Check policy requirements
- Verify not expired
- Confirm complexity rules
- Test password hint

Permission denied:

- Review group assignments
- Check permission inheritance
- Verify user level
- Confirm not Guest user

Account locked:

- Check failed login attempts
- Review policy lockout rules
- Administrator unlock required
- Check LockState property

In this section...