

# AD Groups Quick Reference

## Overview

The IKEA DocuScan application supports three Active Directory groups that are automatically mapped to role claims during authentication.

## AD Groups

AD Group	Role Claim	Purpose	Permissions
ADGroup.Builtin.Reader	Reader	Read-only access	View and search documents only
ADGroup.Builtin.Publisher	Publisher	Document management	Create, edit, register, send documents
ADGroup.Builtin.SuperUser	SuperUser	Full admin access	All operations including delete and user management

## Configuration

**Location:** appsettings.json

```
{  
  "IkeaDocuScan": {  
    "ADGroupReader": "ADGroup.Builtin.Reader",  
    "ADGroupPublisher": "ADGroup.Builtin.Publisher",  
    "ADGroupSuperUser": "ADGroup.Builtin.SuperUser"  
  }  
}
```

**Customize for your organization:**

```
{  
  "IkeaDocuScan": {  
    "ADGroupReader": "IKEA\\DocuScan_Readers",  
    "ADGroupPublisher": "IKEA\\DocuScan_Publishers",  
    "ADGroupSuperUser": "IKEA\\DocuScan_Admins"  
  }  
}
```

## Usage in Code

### Razor Pages/Components

```

@* Restrict entire page to Publishers and SuperUsers *@
@attribute [Authorize(Roles = "Publisher,SuperUser")]

@* Conditional rendering *@
<AuthorizeView Roles="Reader">
    <Authorized>
        <button @onclick="ViewDocument">View</button>
    </Authorized>
</AuthorizeView>

<AuthorizeView Roles="Publisher,SuperUser">
    <Authorized>
        <button @onclick="EditDocument">Edit</button>
    </Authorized>
</AuthorizeView>

<AuthorizeView Roles="SuperUser">
    <Authorized>
        <button @onclick="DeleteDocument">Delete</button>
    </Authorized>
</AuthorizeView>

```

## Code-Behind

```

@inject AuthenticationStateProvider AuthStateProvider

@code {
    private bool canEdit = false;
    private bool canDelete = false;

    protected override async Task OnInitializedAsync()
    {
        var authState = await AuthStateProvider.GetAuthenticationStateAsync();
        var user = authState.User;

        canEdit = user.IsInRole("Publisher") || user.IsInRole("SuperUser");
        canDelete = user.IsInRole("SuperUser");
    }
}

```

## API Endpoints

```

public static void MapDocumentEndpoints(this WebApplication app)
{
    var group = app.MapGroup("/api/documents")
        .RequireAuthorization("HasAccess"); // ALL users with access

    // Read - ALL users (Reader, Publisher, SuperUser)
    group.MapGet("/", GetDocuments);

    // Create/Edit - Publisher and SuperUser only
    group.MapPost("/", CreateDocument)
        .RequireAuthorization(p => p.RequireRole("Publisher", "SuperUser"));

    group.MapPut("/{id}", UpdateDocument)
        .RequireAuthorization(p => p.RequireRole("Publisher", "SuperUser"));

    // Delete - SuperUser only
    group.MapDelete("/{id}", DeleteDocument)
        .RequireAuthorization(p => p.RequireRole("SuperUser"));
}

```

## PowerShell Commands

### Check User's AD Group Membership

```

# View all groups for a user
Get-ADUser -Identity "username" -Properties MemberOf |
    Select-Object -ExpandProperty MemberOf

# Check specific DocuScan groups
Get-ADUser -Identity "username" -Properties MemberOf |
    Select-Object -ExpandProperty MemberOf |
    Where-Object { $_ -like "*DocuScan*" -or $_ -like "*ADGroup.Builtin*" }

```

## Add User to AD Group (Requires Admin Rights)

```

# Add to Reader group
Add-ADGroupMember -Identity "ADGroup.Builtin.Reader" -Members "username"

# Add to Publisher group
Add-ADGroupMember -Identity "ADGroup.Builtin.Publisher" -Members "username"

# Add to SuperUser group
Add-ADGroupMember -Identity "ADGroup.Builtin.SuperUser" -Members "username"

```

## Remove User from AD Group

```

# Remove from Reader group
Remove-ADGroupMember -Identity "ADGroup.Builtin.Reader" -Members "username"

# Remove from Publisher group
Remove-ADGroupMember -Identity "ADGroup.Builtin.Publisher" -Members "username"

# Remove from SuperUser group
Remove-ADGroupMember -Identity "ADGroup.Builtin.SuperUser" -Members "username"

```

## How It Works

1. User authenticates with Windows Authentication
2. WindowsIdentityMiddleware runs
3. Middleware checks if user is in configured AD groups using `WindowsPrincipal.IsInRole()`
4. For each AD group user is in, a corresponding role claim is added
5. Role claims are cached in the authentication ticket for the session
6. Use `@attribute [Authorize(Roles = "...")]` or `User.IsInRole("...")` to check

## Important Notes

### SuperUser Priority

- If user is in **SuperUser AD group**, they automatically get:
  - `IsSuperUser = true` claim (overrides database)
  - `HasAccess = true` claim (overrides database)
  - `ClaimTypes.Role = "SuperUser"`
- SuperUser bypasses **ALL** permission filters
- SuperUser sees **ALL** documents regardless of database permissions

### Multiple Roles

- Users can be in **multiple AD groups** simultaneously
- User will receive **all corresponding role claims**
- Example: User in both Publisher and SuperUser groups gets both role claims

## AD Group Changes

- AD group membership changes may take a few minutes to propagate
- User needs to **log out and log back in** for changes to take effect
- Claims are cached in the authentication ticket

## Development Environment

- On **Linux/WSL**, TestAuthenticationHandler is used (no AD group checks)
- AD group functionality requires **Windows Authentication** on Windows/IIS
- Test AD groups in production or Windows development environment

## Troubleshooting

### User doesn't get role claim

1. Verify AD group membership:

```
Get-ADUser -Identity "username" -Properties MemberOf | Select -ExpandPr  
◀ | ▶
```

2. Check application logs for WindowsIdentityMiddleware entries
3. Verify configuration in appsettings.json matches actual AD group name
4. User may need to log out/in after being added to group

### IsInRole() returns false

1. Try both formats: "GroupName" and "DOMAIN\\GroupName"
2. Ensure Windows Authentication is properly configured
3. IIS application pool identity must have permission to query AD

## Performance concerns

- AD group checks are done **once per authentication**
- Role claims are **cached in the authentication ticket**
- No database queries needed for role checks
- Very efficient for operation-level authorization

## Best Practices

1. Use AD groups for “Can they do this?” (operation authorization)

```
@attribute [Authorize(Roles = "Publisher")]
```

2. Use CurrentUserService for “Which documents?” (data filtering)

```
var currentUser = await CurrentUserService.GetCurrentUserAsync();  
var filtered = documents.Where(d => currentUser.CanAccessDocument(...))  
◀ | ▶
```

3. Prefer AD groups over database flags for role management
  - Easier to manage in centralized AD
  - No database updates needed
  - Leverages existing AD infrastructure

4. Use database permissions for fine-grained filtering
  - Document types
  - Counter parties

- Countries

## Example Scenarios

### Scenario 1: Basic Reader

- **AD Groups:** Reader
- **Database:** No permissions needed (or add for filtering)
- **Access:** View documents, respects database filters if present

### Scenario 2: Department Publisher

- **AD Groups:** Publisher
- **Database:** Permissions for CountryCode = "US", DocumentTypeId = 5
- **Access:** Can create/edit documents, only sees US documents of type 5

### Scenario 3: Global Publisher

- **AD Groups:** Publisher
- **Database:** IsSuperUser = true (or just add SuperUser AD group)
- **Access:** Can create/edit documents, sees ALL documents

### Scenario 4: Administrator

- **AD Groups:** SuperUser
- **Database:** Not required (AD group overrides)
- **Access:** Full admin access, all operations, all documents

## Related Documentation

- Full Authorization Guide: [AUTHORIZATION\\_GUIDE.md](#)
- Database Schema: Migration scripts 04-07 in [/DbMigration/db-scripts/](#)
- Code Reference:
  - WindowsIdentityMiddleware: [/IkeaDocuScan-Web/Middleware/WindowsIdentityMiddleware.cs:127](#)
  - Configuration: [/IkeaDocuScan.Shared/Configuration/IkeaDocuScanOptions.cs:55](#)
  - appsettings.json: [/IkeaDocuScan-Web/appsettings.json:12](#)