Microsoft Exchange

- Restrict App Registration application permission to certain mailboxes
- Share Exchange Resources with external users
- Exchange Contacts Mirroring & Global Distribution Automation

Restrict App Registration application permission to certain mailboxes

> Requirements: Active Exchange Administrator Role and App Registration with application permissions granted.

Application Access Polices

Generally Application Permissions allow an Azure App Registration to access a certain type of data within the whole tenant.

For example the Application Permission Calendars.ReadWrite grants access to every calendar in every mailbox in the Exchange Online environment.

Use case

If, for example, you want to grant an App Registration the Read and Write permission on just 15 calendars, you can do so with an Application Access Policy.

Creating an Application Access Policy

To create an Application Access Policy, you first have to create your App Registration and grant the any Application permission (e.g. Mail.Send).

After that you need to create a Mail Enabled Security Group in the Exchange Admin Center and add the Mailboxes, on which the App Registration' permission shall be activated.

Now you can create the Application Access Policy with the following Command. First you have to log in to Exchange Online.

```
Import-Module ExchangeOnlineManagement
Connect-ExchangeOnline
```

```
New-ApplicationAccessPolicy -AppId "<appregistrationid>" -PolicyScopeGroupId
"<PrimarySMTPAddressofMESG>" -Description "<yourcustomdescription>" -AccessRight Restrict
```

View existing Application Access Policies

If you want to view all existing Application Access Policies in your Tenant, you can do so with this command.

```
Get-ApplicationAccessPolicy
```

For more readability you can view this output in a gridview.

```
Get-ApplicationAccessPolicy | Out-GridView
```

Share Exchange Resources with external users

This guide describes how to share resources from your own environment with external persons (e.g. partners or office members). Internal persons can be invited or adapted in the same way. However, these instructions refer to collaboration outside the company.

Sign in to Exchange Online

The settings are only available via PowerShell and cannot be found in the Exchange Admin Center (WebGUI). Accordingly, you need the PowerShell module “ExchangeOnlineManagement”.

Install PowerShell module

Open a new PowerShell window with administrator rights and execute the installation command:

```
Install-Module -Name ExchangeOnlineManagement -Force
```

Connect to Exchange Online

You must then open a new PowerShell session. This should not have administrator rights and must be started from scratch (do not use an existing window, as the available module paths are not loaded correctly there).

```
Import-Module ExchangeOnlineManagement
Connect-ExchangeOnline
```

A browser pop-up will then appear. Follow the instructions and log in with your administrator account. The user should have Exchange Admin permissions.

This module contains many functions for the Sign In options. You can find more information here:

Connect to Exchange Online PowerShell | Microsoft Learn

Get folder settings for calendar

This command displays all settings. Here you can check the current status of the configuration and your subsequent adjustments.

```
Get-MailboxCalendarFolder -Identity <youremailadress>:\calendar
```

Share calendar state publicly

This command configures the external sharing of calendar availability. The corresponding HTML and iCAL links are then displayed in the overview (execute the previous command again). Look for the parameters "PublishedCalendarUrl" & "PublishedICalUrl".

```
Set-MailboxCalendarFolder -Identity <youremailadress>:\calendar -PublishEnabled $true
```

Set shared details for calendar events

The availabilities are now displayed on the ICAL and HTML accesses by default. More details are not available. You can change the DetailLevel to these three settings so that external persons receive more or less information about the calendar.

- AvailabilityOnly (default)
- LimitedDetails
- FullDetails

```
Set-MailboxCalendarFolder -Identity  <youremailadress>:\calendar -DetailLevel LimitedDetails
```

Allow external calendar booking

With this command you can allow the booking of external mail addresses. If this is executed, all mail requests for resource accounts are answered and a confirmation mail is sent back if available. Take care to only enable this feature on calendars that are intended to be public.

```
Set-CalendarProcessing -Identity <youremailadress> -ProcessExternalMeetingMessages $true
```

Exchange Contacts Mirroring & Global Distribution Automation

Summary

This automation synchronizes contacts between licensed user mailboxes and a shared mailbox folder named **All Contacts** using Microsoft Graph app-only authentication.

It provides:

- One centrally maintained contact set.
- Automatic distribution back to users.
- Traceability in `personalNotes` with sync metadata.

Functional Overview

Step 1: User -> Shared mailbox

- Reads each user's default contacts (`/users/{id}/contacts`).
- Writes into shared mailbox folder **All Contacts**.
- Also stamps metadata back to source user contacts when needed.

Step 2: Shared mailbox -> User

- Reads shared mailbox contacts only from folder **All Contacts**.
- Syncs them into each user's default contacts.

Matching and deduplication

Contacts are matched in this order:

1. `SyncId` in `personalNotes`
2. Primary email (`emailAddresses[0].address`)
3. `displayName` (fallback)

Metadata Stamping (`personalNotes`)

The script maintains sync tags in `personalNotes`:

- `SyncId=<guid>`
- `CreatedBy=<upn-or-id>`
- `LastUpdatedAt=<utc-iso8601>`
- `LastUpdatedBy=<upn-or-id>`

Notes behavior:

- Existing non-sync note text is preserved.
- Old sync tags are replaced with fresh values.
- `SyncId` is generated if missing.
- `CreatedBy` is preserved from existing synced contact when available.

Update Behavior

`UpdateExisting` is enabled by default.

An existing target contact is only updated if:

- Data has changed, **and**
- Target contact is older than source (`lastModifiedDateTime` comparison).

If no change or target is newer/equal, it is skipped.

User Scope / Test Mode

User list resolution:

1. Per-tenant test users (if configured).
2. Otherwise all licensed users from Graph:
   - `/users?$filter=assignedLicenses/any(x:x/skuId ne null)`

Shared mailbox account is excluded from user sync loops.

Configuration

Preferred: full JSON config

Use `AUTOMATION_SYNCCONTACTS_CONFIG_JSON`:

```
{
  "MicrosoftTenants": [
    {
      "TenantId": "tenant-id-or-domain",
      "ClientId": "app-client-id",
      "ClientSecret": "app-client-secret",
      "GlobalAddressBookUserId": "shared.contacts@contoso.com",
      "TestUserList": ["user1@contoso.com"]
    }
  ],
  "DryRun": "true"
}
```

Multi-tenant via separate env var

Use `AUTOMATION_SYNCCONTACTS_MICROSOFT_TENANTS_JSON` with tenant array.

Backward-compatible single-tenant env vars

- `AUTOMATION_SYNCCONTACTS_TENANT_ID`

- `AUTOMATION_SYNCCONTACTS_CLIENT_ID`
- `AUTOMATION_SYNCCONTACTS_CLIENT_SECRET`
- `AUTOMATION_SYNCCONTACTS_GLOBAL_ADDRESS_BOOK_USER_ID`
- `AUTOMATION_SYNCCONTACTS_DRY_RUN` (`true/false`)

Optional tenant test-user map env var

Works even with full config JSON:

- `AUTOMATION_SYNCCONTACTS_TEST_USER_LIST_BY_TENANT_JSON`

Example:

```
{
  "contoso.onmicrosoft.com": ["user1@contoso.com", "user2@contoso.com"],
  "fabrikam.onmicrosoft.com": ["user3@fabrikam.com"]
}
```

Lookup keys include tenant identifiers like `TenantId`, `PrimaryDomain`, `TenantDomain`.

Prerequisites

- Shared mailbox exists and is reachable as `GlobalAddressBookUserId`.
- Azure AD app registration with application permissions.
- Admin consent granted.
- App-only access to mailbox contacts.

Required Microsoft Graph Permissions

Application permissions:

- `Contacts.ReadWrite`
- `User.Read.All` or `Directory.Read.All`

Dry Run

Set `DryRun` / `AUTOMATION_SYNCCONTACTS_DRY_RUN` to `true` to simulate actions without writes. The script logs what would be created/updated and prints creation summaries.

CI/CD

Designed for automation pipelines (for example GitLab CI/CD). Store secrets as protected CI/CD variables.

Troubleshooting

- **No tenants configured**: set `AUTOMATION_SYNCCONTACTS_CONFIG_JSON` or `AUTOMATION_SYNCCONTACTS_MICROSOFT_TENANTS_JSON`.
- **Insufficient privileges**: missing admin consent or Graph permissions.
- **Shared mailbox not found**: incorrect `GlobalAddressBookUserId`.
- **No licensed users found**: tenant has no users matching license filter.
- **Config JSON parse errors**: invalid JSON in env vars.

Security Notes

- Keep `ClientSecret` in secure secret storage.
- Restrict app permissions to minimum required.
- Limit and audit access to shared mailbox contact data.

Script Source

Automation-ContactsMirroring.ps1