# Alarm Email Example (10.1.5+ workflow)

Notify users by email — 10.1.5+ workflow with bundled OAuth2 libraries.



**Version 10.1.5+**

---

Starting with FrameworX 10.1.5, the SMTP-over-OAuth2 libraries (`MailKit`, `MimeKit`, `BouncyCastle.Cryptography`, `Google.Apis.Auth`) ship with the product by default. The separate DLL download (`Email.rar`) required by the 10.1.4 page is no longer needed. For 10.1.4 behavior, see the parent page.

This solution demonstrates how to send alarm notifications by email using OAuth2 authentication against Gmail or Microsoft 365 (Outlook). All required libraries ship with FrameworX 10.1.5 — the only preparation step is obtaining OAuth2 credentials from your provider.

---

## Technical Information

The method *`AlarmEvents(AlarmEventInfo[] events)`* is automatically called whenever a conditional alarm event occurs. It receives an array of events and processes the first one to send a notification.

### Required Libraries (shipped by default in 10.1.5)

The following libraries are available to server-side scripts out of the box — no `Email.rar` download, no manual DLL copy:

- `MailKit` — SMTP client with OAuth2 XOAUTH2 SASL support
- `MimeKit` — MIME message construction
- `BouncyCastle.Cryptography` — cryptography primitives used by MimeKit
- `Google.Apis.Auth` — Google OAuth2 flow for Gmail
- `Microsoft.Identity.Client` (MSAL) — Microsoft OAuth2 flow for Outlook and Microsoft 365

### Alarm Script — Direct MailKit Pattern

Add the following namespace declarations to the Script Task (in the *NamespaceDeclarations* field, not the code body):

```
MimeKit;MailKit.Net.Smtp;MailKit.Security
```

Script Task body:

```
public void AlarmEvents(AlarmEventInfo[] events)
{
    if (events == null || events.Length == 0)
        return;

    AlarmEventInfo ev = events[0];
    @Info.Trace("Alarm event state = " + ev.State.ToString());

    // Only send for newly-active alarms
    if (ev.State != 1)
        return;

    string body =
        "Time: "    + ev.ActiveLocalTime.ToString() + "\n" +
        "Message: " + ev.Message                    + "\n" +
        "Area: "    + ev.Area                       + "\n" +
        "Group: "   + ev.Group                      + "\n" +
        "Tag: "     + ev.TagName                    + "\n";

    // Build the MIME message
    var msg = new MimeMessage();
    msg.From.Add(new MailboxAddress("FrameworX Alarms", @Tag.fromEmail));
    msg.To.Add(new MailboxAddress("Operator", @Tag.toEmail));
    msg.Subject = "Alarm: " + ev.Message;
    msg.Body = new TextPart("plain") { Text = body };

    // Acquire OAuth2 bearer token (see 'Credentials' section for provider setup)
    string bearer = GetOAuth2AccessToken();

    // Send via SMTP with XOAUTH2 SASL
    using (var client = new SmtpClient())
    {
        client.Connect("smtp.gmail.com", 587, SecureSocketOptions.StartTls);
        client.Authenticate(new SaslMechanismOAuth2(@Tag.fromEmail, bearer));
        client.Send(msg);
        client.Disconnect(true);
    }

    @Info.Trace("Alarm email sent to " + @Tag.toEmail);
}
```

The `GetOAuth2AccessToken()` helper belongs in a reusable Script Class and wraps your provider's token flow — see the *Credentials* section below for Gmail and Outlook specifics. For Outlook / Microsoft 365 replace the SMTP host with `smtp.office365.com`.

---

## Credentials

OAuth2 replaces username/password authentication for Gmail and Outlook. You need to register an OAuth2 application with your provider and obtain client credentials before the script can acquire access tokens.

### For Gmail

To authenticate with Gmail via OAuth2, generate a **Client ID** and **Client Secret** through the Google Cloud Console:

- Go to Google Cloud Console
- Create or select a project
- In **APIs & Services > Library**, search for and enable the **Gmail API**
- In **OAuth consent screen**, choose *External*, then set app name, support email, and developer contact info
- In **Credentials**, click **Create Credentials > OAuth Client ID** and choose **Desktop App**
- After creating, you'll receive your **Client ID** and **Client Secret**
- Back in **OAuth consent screen**, scroll to **Test users** and add your own Gmail address (required while the app is in testing)

Use the credentials in your token-acquisition helper — typical pattern with `Google.Apis.Auth`:

```
// Stored as FrameworX tags or Secrets - never hardcode in production
string clientId     = "your-client-id.apps.googleusercontent.com";
string clientSecret = "your-client-secret";
```

### For Outlook / Microsoft 365

To authenticate with Outlook using OAuth2, register an application in the Azure portal:

- Go to *Azure portal > Microsoft Entra ID > App registrations*
- Register a new application and note the **Application (client) ID** and **Directory (tenant) ID**
- Define redirect URIs appropriate to your flow (typically `http://localhost` for desktop apps)
- Under **API permissions**, grant Microsoft Graph `Mail.Send` (or the legacy SMTP `SMTP.Send` scope, depending on tenant policy)
- Use the MSAL library (`Microsoft.Identity.Client`) to acquire tokens

Point the MailKit `SmtpClient` at `smtp.office365.com` on port 587 with `SecureSocketOptions.StartTls` and the same `SaslMechanismOAuth2` pattern as the Gmail example.

---

## Security Notes

- **Never hardcode client secrets in scripts.** Store them as FrameworX tags, use the `Secrets` module, or read from an OS-level secret store.
- **OAuth2 tokens expire.** Cache the access token and its expiration; refresh before sending when the TTL is near zero. Do not acquire a new token on every alarm.
- **For on-premise SMTP relays** (corporate Exchange, Postfix, plant mail gateway), OAuth2 is often not required — use basic authentication or anonymous relay via the same MailKit `SmtpClient`, just without `SaslMechanismOAuth2`. This is the typical scenario for industrial sites without internet access to Gmail or Microsoft 365.

---