TECHNISCHE LÖSUNGSBESCHREIBUNG OFFICEMASTER WEBAPI-KOMPONENTE

Ferrari electronic AG
Ruhlsdorfer Straße 138
14513 Teltow

info@ferrari-electronic.de
Tel. +49 3328 45590
Fax +49 3328 455960
https://ferrari-electronic.de

Inhaltsverzeichnis

- Rechtliche Informationen 6
- Versionshistorie und Impressum 7

1. Überblick

- 1.1. Einführung 9
- 1.2. OfficeMaster Suite 10
- 1.3. Vorhandene Komponenten zur Software-Integration 11
- 1.4. WebAPI-Komponente 12

2. Netzwerk-Architektur

- 2.1. OfficeMaster Gatekeeper als Reverse-Proxy 14
- 2.2. Firewalling und Port-Forwarding für externen Zugriff 15

3. API der WebAPI-Komponente

- 3.1. Beschreibung 17

- 3.2. Einrichtung 18
- 3.3. Zugriff zur Dokumentation 20

4. Nutzung mit Microsoft Dynamics 365

- 4.1. Überblick 22
- 4.2. Callback-Flow 26
- 4.3. Send-Flow 29
- 4.4. Nutzungsbeispiel: Ferrari electronic AG 33

5. Go Code-Beispiel zur Nutzung der Web-API

- 5.1. Hinweis 35
- 5.2. Go Code 36

Rechtliche Informationen

Version 1.1

9. Juli 2024 | Ferrari electronic AG

UNIFIED MESSAGING

www.ferrari-electronic.de

Die in diesem Dokument enthaltenen Informationen wurden mit größter Sorgfalt zusammengestellt. Dennoch können fehlerhafte Angaben nicht völlig ausgeschlossen werden. Die Ferrari electronic AG haftet nicht für eventuelle Fehler und deren Folgen. Hinweise und Kommentare richten Sie bitte an:

info@ferrari-electronic.de

Versionshistorie und Impressum

Revision	Datum	Änderungen
1.1	9.7.2024	Kapitel Nutzung mit Microsoft Dynamics 365
1.0	8.4.2024	Datenimport, initialer Text

Herausgeber:
Ferrari electronic AG
Ruhlsdorfer Str. 138
(DE) 14513 Teltow
Internet:
www.ferrari-electronic.de
Telefon:
+49 3328 455 90
Fax:
+49 3328 455 960
E-Mail:
info@ferrari-electronic.de
Autoren:
M. With, J. Kühner, R. Fiedler

1. Überblick

1.1. Einführung

Dieses Dokument beschreibt den Funktionsumfang und die Schritte zur Nutzung der WebAPI-Komponente der OfficeMaster Suite. Die Lösungsbeschreibung richtet sich an Entscheider und IT-Verantwortliche. Die enthaltenen Informationen dienen dazu, die für eine Integration notwendigen Schritte und Maßnahmen vor einer Kaufentscheidung abwägen zu können.

Der Haupteinsatzfall der WebAPI-Komponente ist die Integration der OfficeMaster Suite in Cloud-Lösungen. Bei lokal ausgeführter Software kann alternativ die Dateischnittstelle (File-Gateway) genutzt werden. Je nach anzubindender Software-Lösung kann der Aufwand für die Dateischnittstelle oder die Web-API geringer sein.

Obwohl auch Beispiel-Code angegeben ist, sollte es bei den meisten Cloud-basierten Lösungen möglich sein, durch entsprechende Anpassung von Abläufen mit Low-Code-Konfigurationsoberflächen die Web-API ansteuern zu können.

1.2. OfficeMaster Suite

Die OfficeMaster Suite bietet die Funktionen

- Fax und NGDX (Next Generation Document Exchange, PDF-Übertragung per Fax),
- SMS,
- Voicemail,
- EPost-Brief und
- XRechnung.

über verschiedene Konnektoren zur Integration in Softwaresysteme einer Organisation an. Spezielle Konnektoren werden für verschiedene Softwaresysteme angeboten:

- Microsoft Exchange (On-Premises per EWS),
- Microsoft Exchange Online (Graph-API),
- allgemeine E-Mailsysteme (Mail-Gateway, SMTP und LDAP),
- Lotus/IBM/HCL Notes,
- SAP R/3, ERP und S4/HANA

Darüber hinaus gibt es Komponenten, welche der Integration in andere Softwaresysteme dienen.

1.3. Vorhandene Komponenten zur Software-Integration

Für die Integration des Dokumentenaustauschs stehen seit vielen Jahren eine Reihe von Schnittstellen zur Verfügung:

- LPD-Schnittstelle zur Beauftragung als Druckjob,
- File-Gateway zur Beauftragung per Dateischnittstlle (Hot-Folder) bzw. zur Übergabe in Empfangsordner,
- FOFI (File-Out/File-In) zur Einbindung kundenspezifischer Bearbeitungsschritte,
- Sign-Komponente zur Einbindung digitaler Signatur.

Vor dem Release 8 der OfficeMaster Suite gab es jedoch keine Möglichkeit, Sendeaufträge per REST-API an die OfficeMaster Suite zu senden. Dies ist für die Integration in cloudgestützte Lösungen notwendig.

1.4. WebAPI-Komponente

1.4.1. Funktion

Bei Software-Integrationen wird das File-Gateway mit Dateischnittstelle häufig eingesetzt. Wenn Software allerdings in der Cloud ausgeführt wird, ist diese Schnittstelle nicht mehr gut nutzbar. Die WebAPI-Komponente übernimmt diese Funktion in diesen Fällen.

Die WebAPI-Komponente erlaubt es,

- neue Sendeaufträge an die OfficeMaster Suite zu übermitteln und
- Nummern für den Empfang zu registrieren und dabei eine Ziel-URI für den Empfang zu definieren.

Mit diesen Mitteln ist der Empfang und Versand von Aufträgen (Fax, SMS, EPost-Brief, ...) möglich.

1.4.2. Einrichtung der Komponente

Die Komponente kann im *OfficeMaster Admin Center* oder der *Messaging Server Konfiguration* hinzugefügt werden. Hier können auch die Einstellungen zum Zugriff auf die WebAPI-Komponente verändert werden. In der Standard-Konfiguration bindet sich die WebAPI-Komponente auf die localhost-Adresse und ist nur über den Reverse-Proxy im Gatekeeper erreichbar. Mittels der manuellen Konfiguration kann hier eine von außen erreichbare IP-Adresse und ein abweichender Port konfiguriert werden.

1.4.3. Authentisierung einer Client-Anwendung

Der Zugriff auf die WebAPI-Komponente wird per API-Keys authentisiert. Neue API-Keys können auf der Konfigurationsseite der Komponente hinzugefügt werden. Dabei können ein Ablaufdatum und Zugriffsrechte (Senden, Empfangen, Status) angegeben werden.

Der API-Key sollte dann bei der API-Nutzung durch die Client-Anwendung angegeben werden.

2. Netzwerk-Architektur

2.1. OfficeMaster Gatekeeper als Reverse-Proxy

Die WebAPI-Komponente kann auf dem OfficeMaster Suite Server in einer von zwei Betriebsarten eingesetzt werden:

- Entweder die Komponente bindet ein TCP-Port auf der localhost-Adresse und ist von außerhalb der Maschine nur über den Reverse-Proxy des Gatekeepers erreichbar.
- Oder die WebAPI-Komponente bindet nach dem Start eine konfigurierbare Adresse (IP-Nummer/Port) und ist dann unter dieser direkt erreichbar.

Die erste Variante hat den Vorteil, dass keine Konfiguration nötig ist und alle Funktionen der OfficeMaster Suite unter der Adresse des Gatekeepers erreichbar sind. Der Nachteil ist, dass alle Funktionen des Gatekeepers, die auch von den Konfigurationsprogrammen der OfficeMaster-Suite genutzt werden, erreichbar sind und man unter Umständen nur die API der WebAPI-Komponente per Port-Forwarding von außen erreichbar haben möchte.

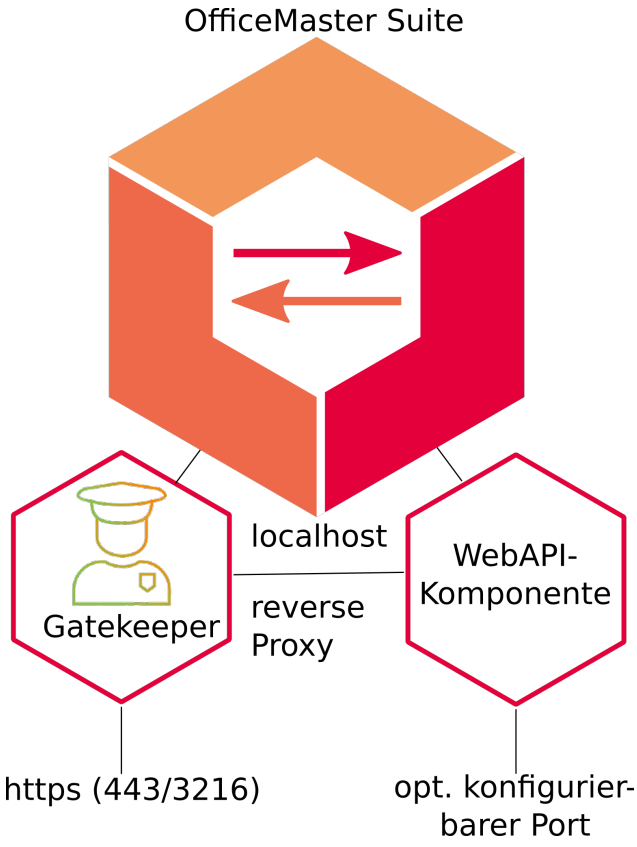


Abbildung 2.1: Nutzung eines Reverse Proxies

2.2. Firewalling und Port-Forwarding für externen Zugriff

Je nach Netztopologie der Organisation, in der die WebAPI-Komponente eingesetzt wird, kommen Firewalls zum Einsatz. Um die WebAPI-Komponente von z.B. einem Cloud-System erreichbar zu machen, müssen entsprechende Port-Forwarding-Regeln in den Firewalls angelegt werden. Aus Sicherheitsgründen kann es dabei sinnvoll sein, die Port-Forwardings nur von bestimmten Quell-IP-Adressen zu erlauben.

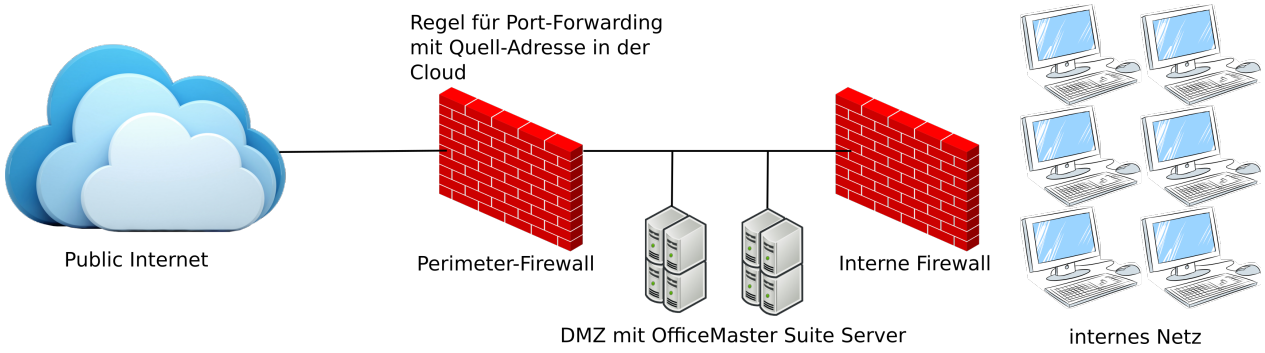


Abbildung 2.2: Beispielhafte Netzwerk-Architektur

3. API der WebAPI-Komponente

3.1. Beschreibung

Die WebAPI-Komponente stellt eine API via HTTPS bereit. Über diese Schnittstelle können Aufträge (Jobs) versendet und empfangen werden. Dies ermöglicht einer beliebigen Software beispielsweise Rechnungen und Mitteilungen mit der OfficeMaster Suite zu übertragen.

Statusrückmeldungen erfolgen zweistufig. Die erste Stufe stellt die Antwort beim Absenden des Auftrags an die Web-API dar. Ein fehlerhafter Auftrag kann an dieser Stelle bereits von der Web-API abgelehnt werden, wenn beispielsweise erforderliche Felder im Auftrag fehlen. Dies äußert sich durch einen HTTP-Statuscode ungleich 200 (OK). Die zweite Stufe stellt eine Rückmelde-URL dar. Diese kann bei der Beauftragung angegeben werden. Die Web-API meldet den Status des Auftrages mittels HTTPS an die angegebenen Rückmelde-URLs, egal ob fehlerhaft oder erfolgreich.

3.2. Einrichtung

Information!
Eine genauere Beschreibung der einzelnen Konfigurationsparameter finden Sie im Konfigurationsprogramm mit aktivierter Hilfe.

3.2.1. Versand

Für den Versand von Dokumenten wird ein API-Key benötigt. Dieser kann über die Konfigurationsoberfläche der WebAPI-Komponente im Konfigurationsprogramm angelegt werden. Die Berechtigung “Versand” für den API-Key ist notwendig um Dokumente zu versenden.

Der Endpoint für den Versand lautet `https://{host}:3216/webapi/v2/transfer`. Ein Beispiel für den Versand von Dokumenten finden Sie im Abschnitt 5.2.

3.2.2. Empfang

Um Dokumente für eine oder mehrere Rufnummern zu erhalten, muss sich dafür registriert werden. Dies kann über die API als dynamische Registrierung oder per Konfiguration als statische Registrierung erfolgen.

Dynamische Registrierung

Eine dynamische Registrierung erfolgt über die API-Schnittstelle. Dort wird angegeben, für welche Rufnummern Empfangsmitteilungen versendet werden sollen, wobei es möglich ist, mehrere Empfangs-URLs einzutragen. Auch hier ist ein API-Key mit entsprechender Empfangsberechtigung notwendig. Im Gegensatz zu der statischen Registrierung haben dynamische Registrierungen keine unbegrenzte Lebensdauer, sie werden nach einer konfigurierbaren Zeit ungültig. Es ist also in regelmäßigen Abständen notwendig die Registrierung zu wiederholen, um die Lebensdauer wieder zurückzusetzen. Bei einem Serverausfall werden somit unzustellbare Empfangsmitteilungen vermieden.

Der Endpoint für die dynamische Registrierung von Empfangsmitteilungen lautet `https://{host}:3216/webapi/v2/receive`.

Statische Registrierung

Für die Anbindung an ein CRM ist die statische Registrierung angemessen. Einträge müssen hier über das Konfigurationsprogramm vorgenommen werden und bleiben unbegrenzt gültig. Bei einer Fehlzustellung wird die Verbindung in regelmäßigen Abständen erneut versucht, sodass Mitteilungen nur im Ausnahmefall unzustellbar sind. Eine Anpassung dieser Einträge ist nicht während der Laufzeit möglich, sondern bedarf eines Neustarts der WebAPI-Komponente. Es wird kein API-Key benötigt, da für diesen Registrierungstyp kein API-Aufruf abgesetzt werden muss.

3.3. Zugriff zur Dokumentation

Die Dokumentation der gesamten API-Schnittstelle ist unter der Route `https://{host}:3216/webapi/v2/doc/` verfügbar. Die API-Dokumentation ist interaktiv und kann auch zum Ausprobieren der API-Aufrufe genutzt werden.

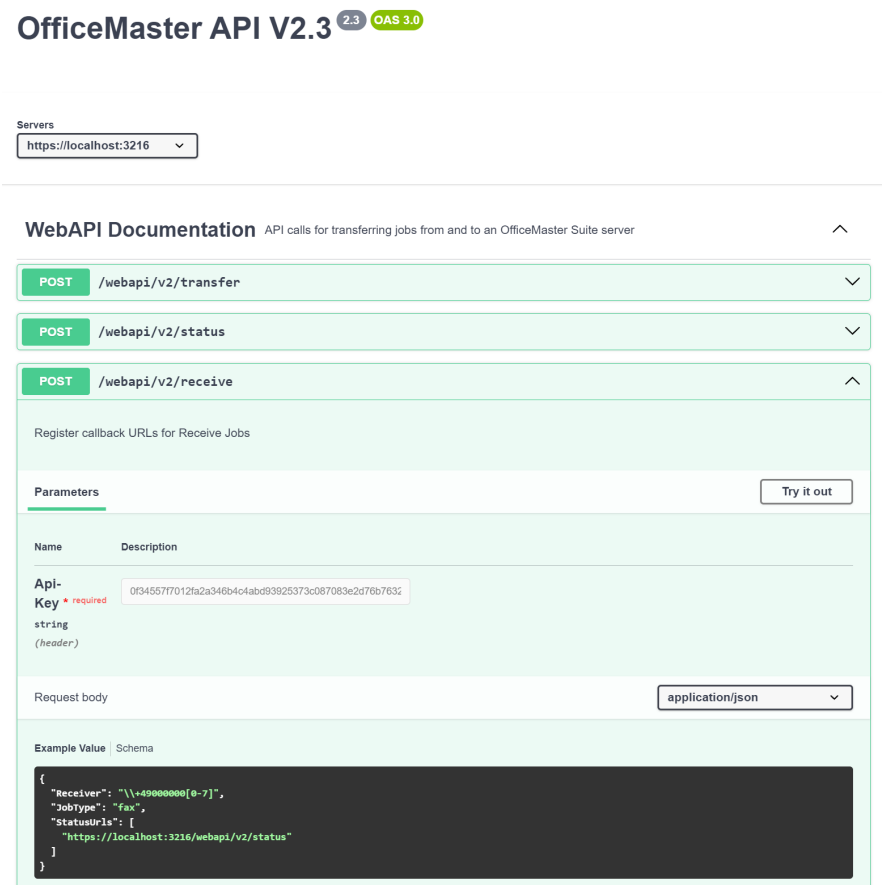


Abbildung 3.1: interaktive API-Dokumentation

Die Dokumentation für die Konfiguration der WebAPI-Komponente finden Sie in der Online-Hilfe in der Messaging Server Konfiguration oder im OfficeMaster Admin Center.

Die API kann auch zur Einbindung in Low-Code-Umgebungen genutzt werden. Ein Beispiel dafür ist die Einbindung in Dynamics 365.

4. Nutzung mit Microsoft Dynamics 365

4.1. Überblick

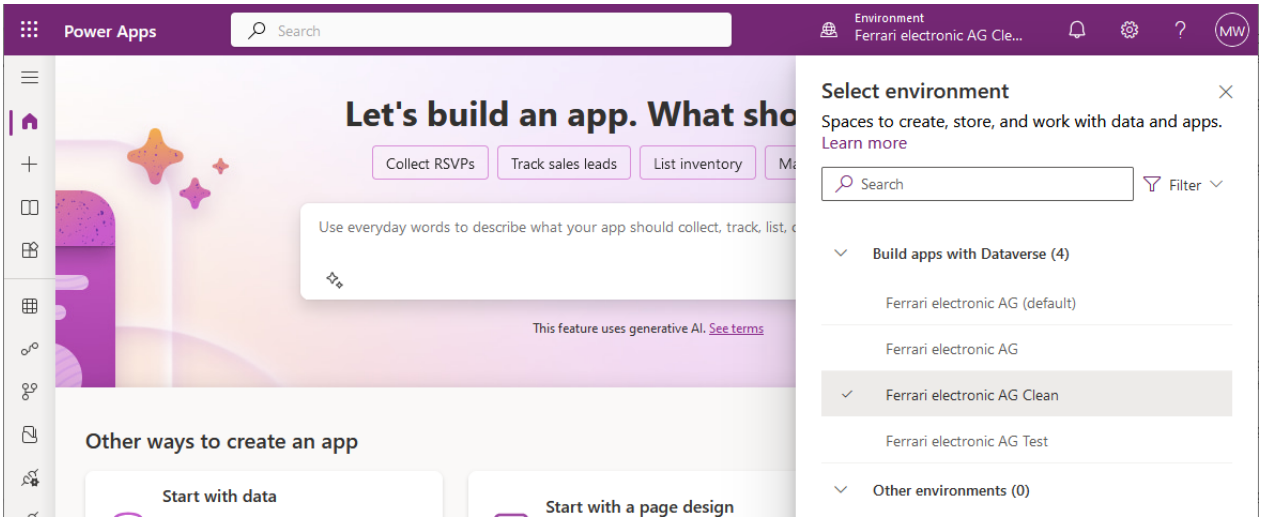
Um die WebAPI-Komponente in Dynamics 365 für den Versand zu benutzen, gibt es die Möglichkeit zwei *PowerAutomate*-Flows zu erstellen: Einen Send-Flow und einen Callback-Flow. Flows bestehen aus von Microsoft bereitgestellten Triggern (Auslösern) und Aktionen und können auf die Datenbank des CRM-Systems zugreifen und auf HTTP-POST-Requests reagieren.

Zudem ist es nötig, einen API-Key in der WebAPI-Komponente zu erzeugen und diesen bei Anfragen an die WebAPI-Komponente zu verwenden. Zusätzliche Informationen dazu findet man im Handbuch der OfficeMaster Suite im Bereich “Web-API” → “API-Keys”.

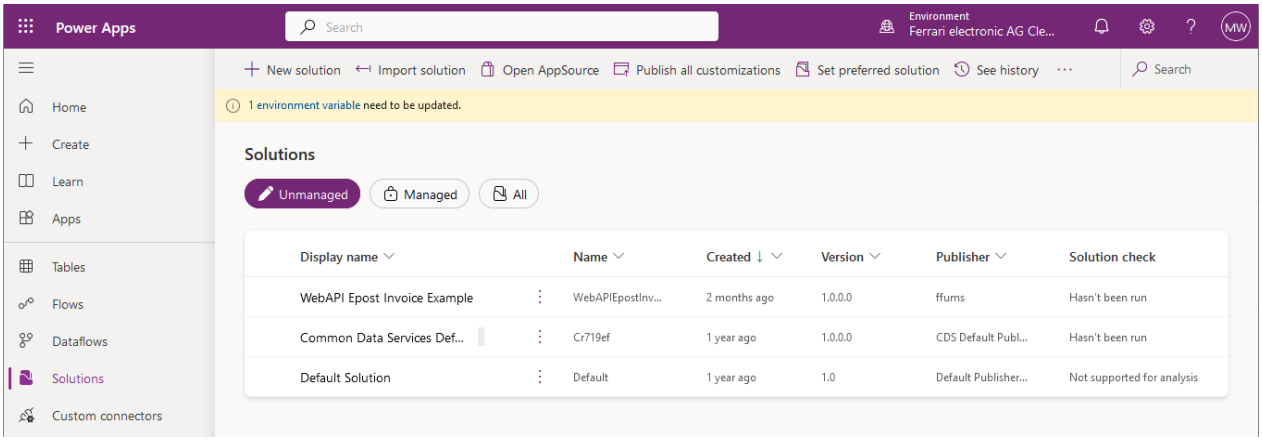
Die Reihenfolge der Ausführung der Flows (*Send-Flow*, *Callback-Flow*) ist abweichend von der Reihenfolge der Einrichtung der Flows. Dies liegt daran, dass die URL des Callback-Flows vom CRM bestimmt wird und bei der Einrichtung des Send-Flows angegeben werden muss. Die Flows werden hier in der Reihenfolge der Einrichtung beschrieben.

4.1.1. Solution erstellen

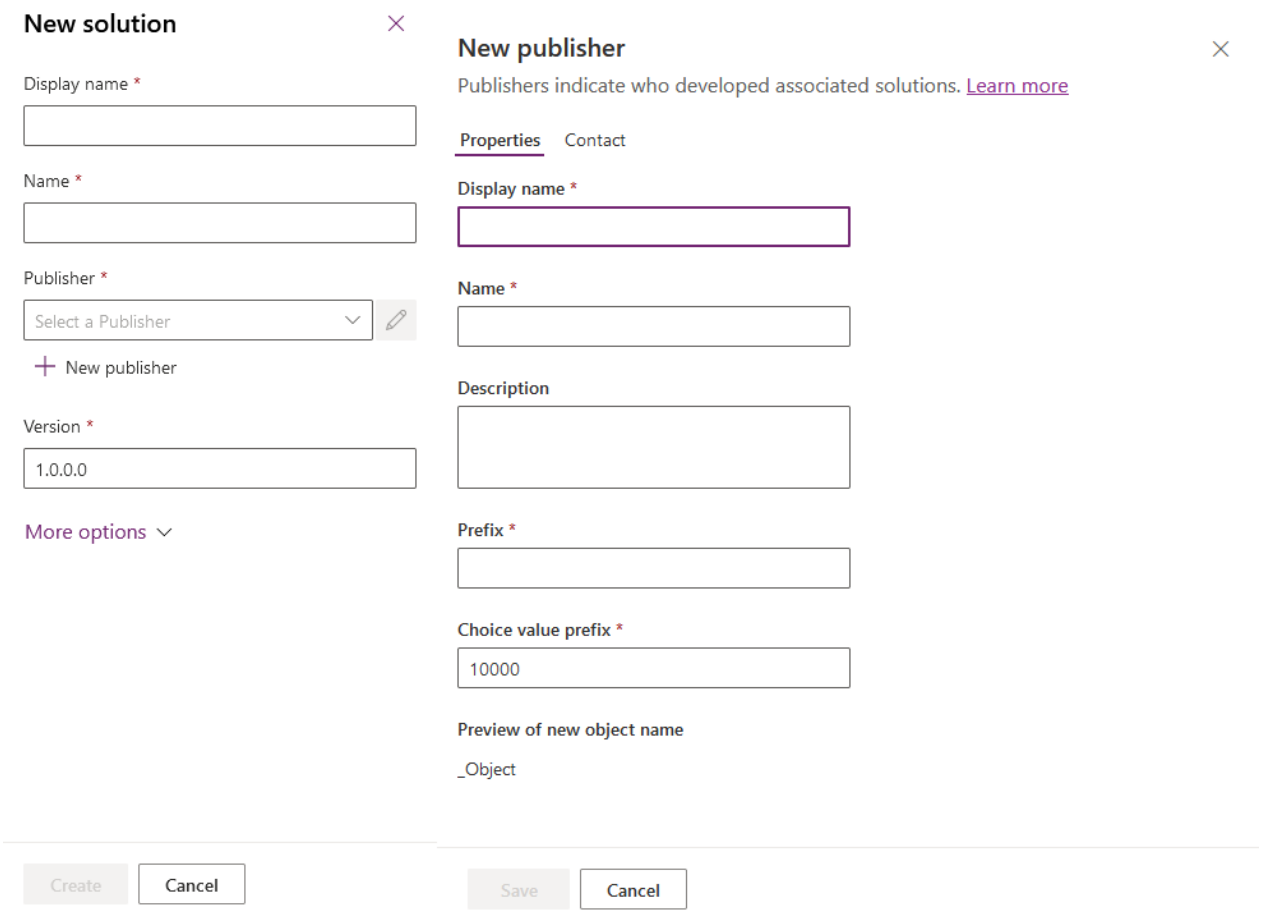
Um einen Flow zu erstellen öffnet man `https://make.powerapps.com`, und wählt rechts oben das “Environment” aus.



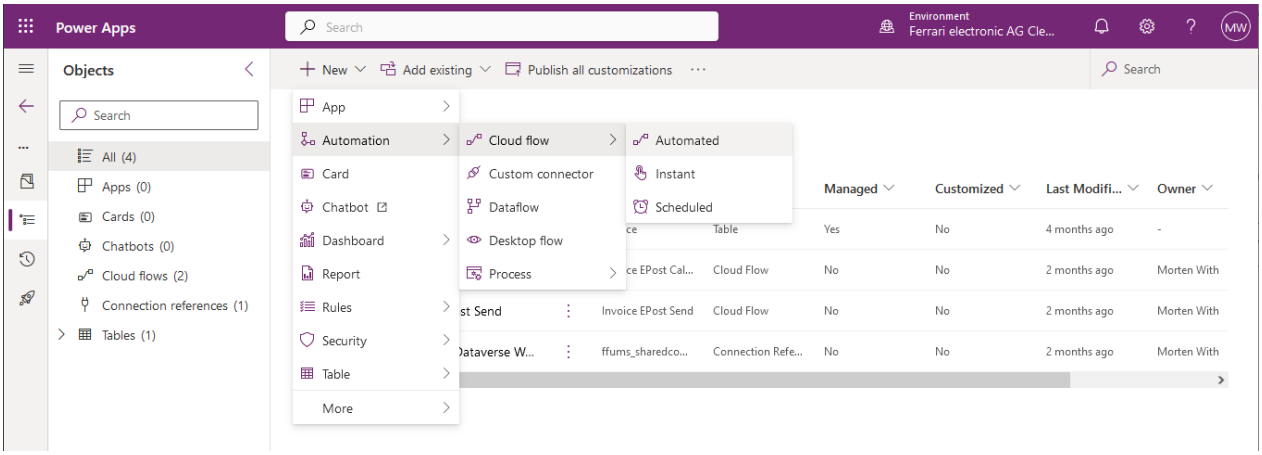
Anschließend öffnet man die Solutions Übersicht, da Flows immer in einer Solution erstellt werden sollten, falls ein Nutzer die Organisation verlässt. Falls noch kein Publisher außer dem “CDS Default Publisher” existiert, bietet es sich an einen neuen für die eigenen Änderungen anzulegen.



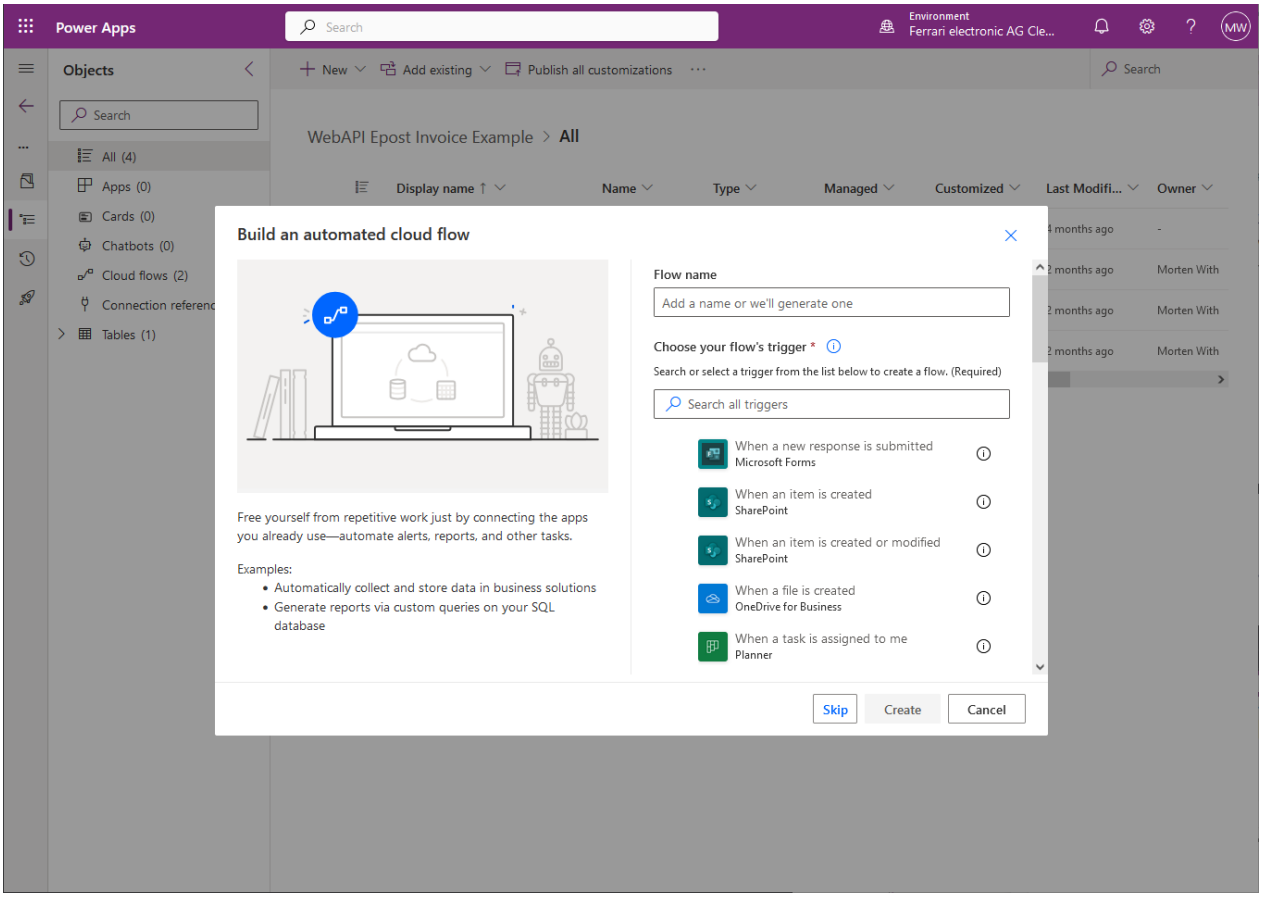
Der “New publisher” Dialog ist im “New solution” Dialog, alternativ wählt man einen existierenden aus.

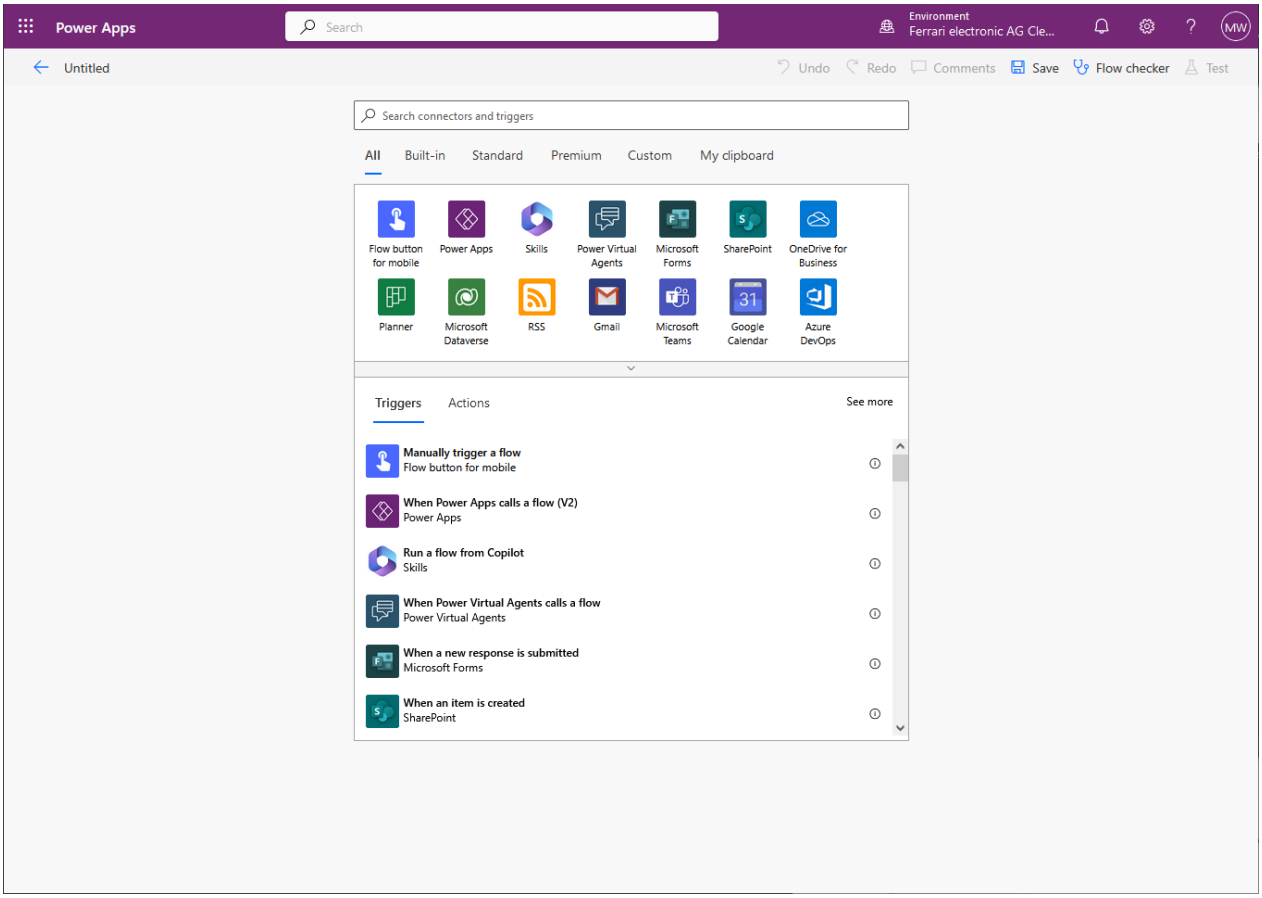


4.1.2. Flow erstellen



Nun kann man einen Flow erstellen. Wenn man im ersten Dialog auf “Skip” drückt, hat man mehr Trigger-Auswahlmöglichkeiten.





“Microsoft Dataverse” reagiert auf Datenbankänderungen, z.B. für den Send-Flow. Wenn man nach HTTP sucht, findet man HTTP Trigger, z.B. für den Callback-Flow.

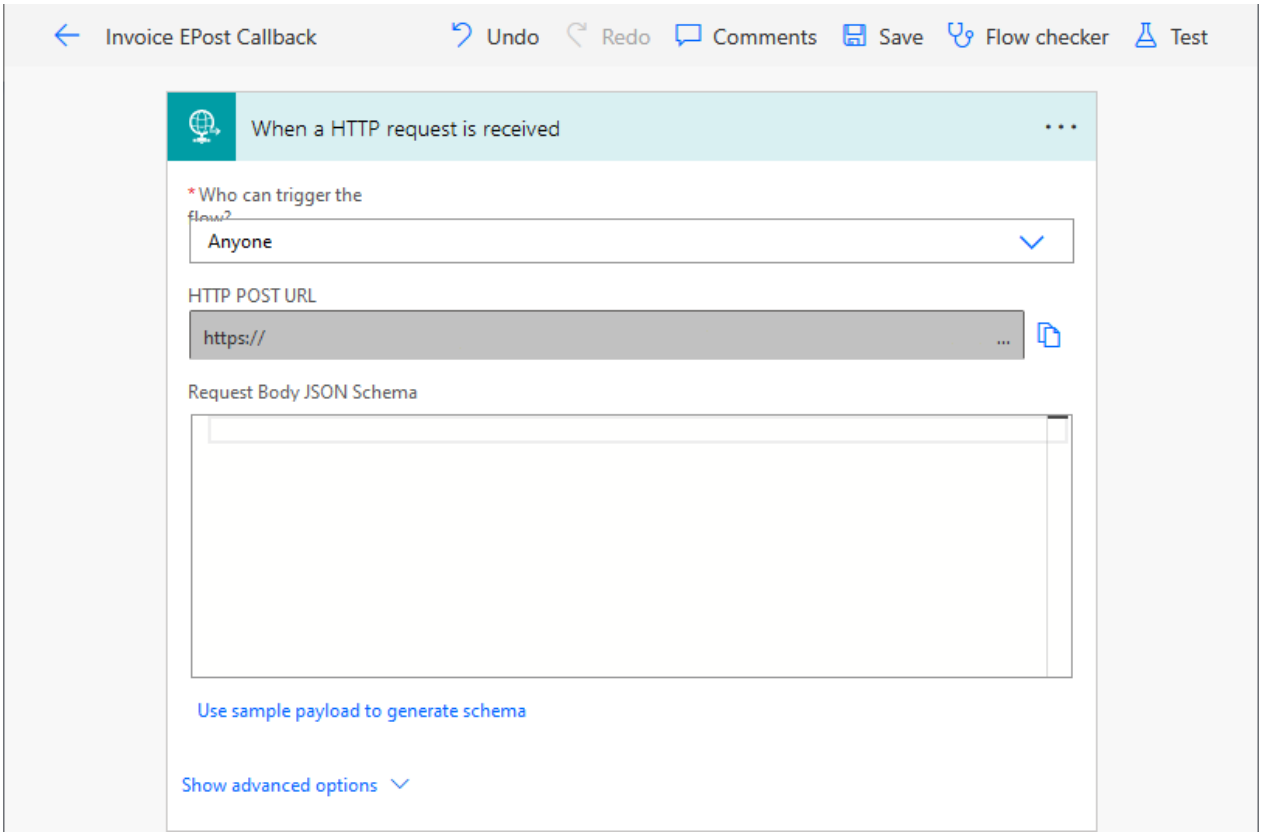
4.2. Callback-Flow

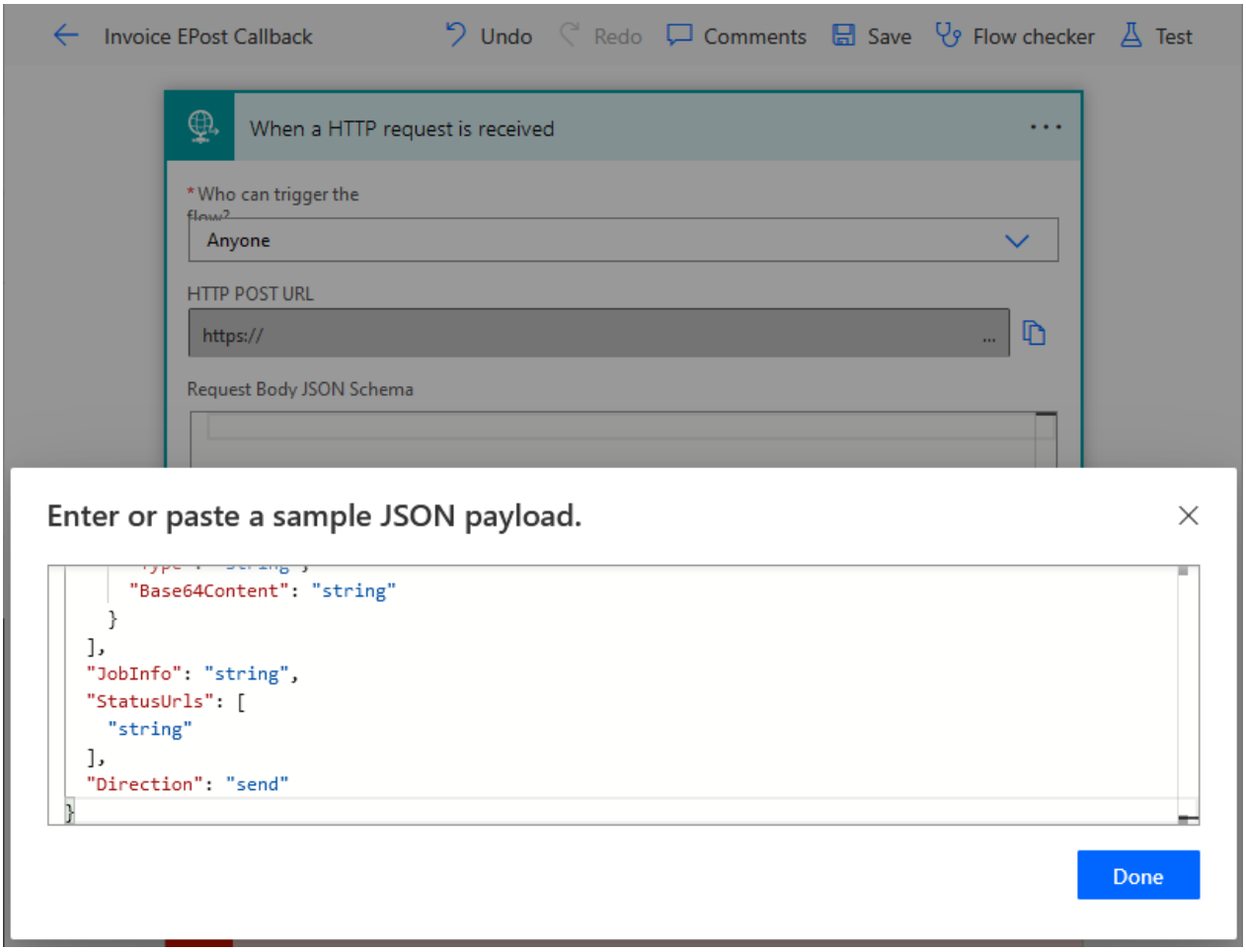
Der Callback-Flow reagiert auf HTTP-POST an eine für diesen Flow generierte URL und muss zuerst erstellt werden, damit dessen URL im Send-Flow angegeben werden kann. Im Body eines vom Callback-Flow empfangenen HTTP-POST-Requests befindet sich eine *JobInfo*-Struktur. Diese ist weiter unten im Abschnitt *Send-Flow* erläutert.

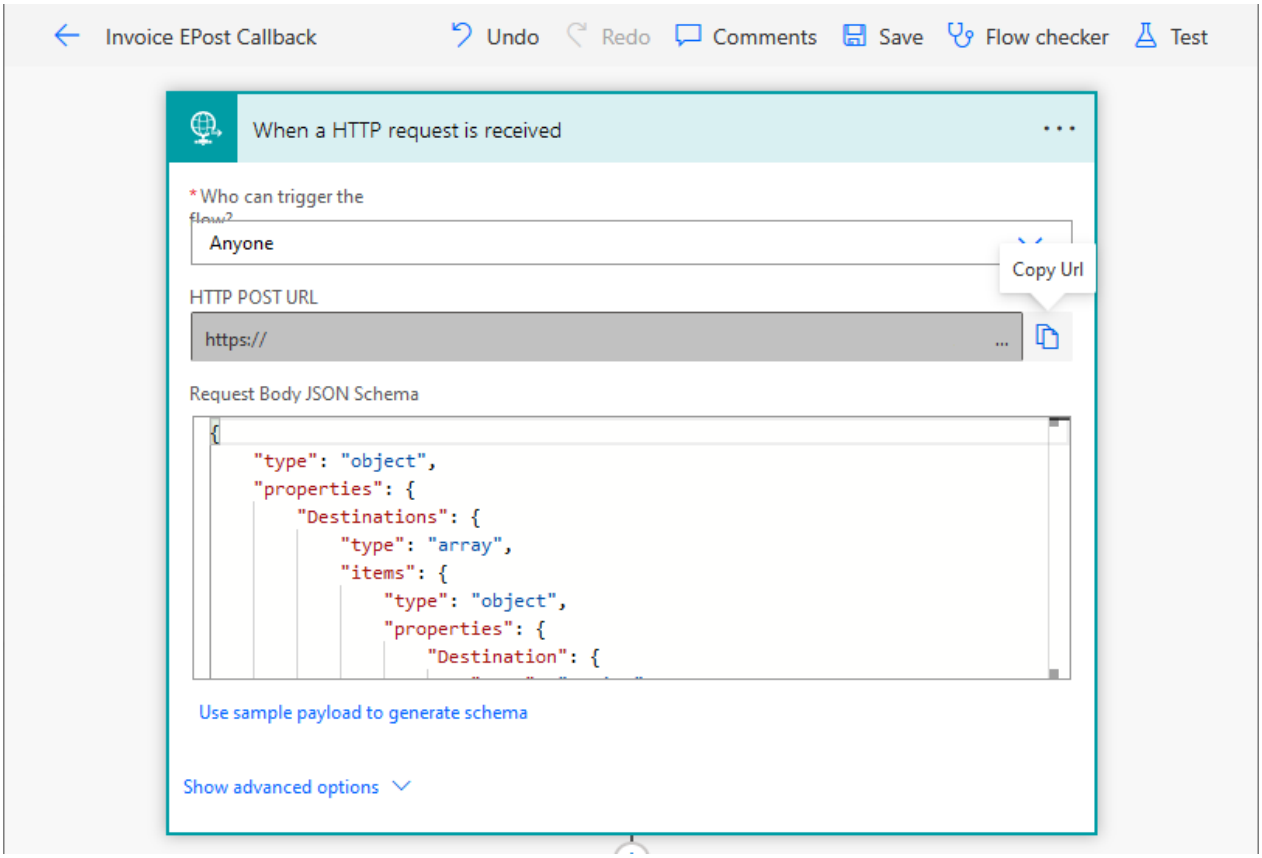
Der Flow dient dazu, die Statusrückmeldung der WebAPI-Komponente auf Fehlerfreiheit zu prüfen und markiert das Dokument als versendet. Alternativ findet eine Fehlerbehandlung statt (z.B. Benachrichtigung des Nutzers).

Es wäre theoretisch möglich den gleichen Callback-Flow für verschiedene Send-Flows zu verwenden, aber je nach z.B. unterschiedlicher Fehlerbehandlung kann dies unübersichtlich werden, insofern sollte pro Send-Flow ein Callback-Flow erstellt werden.

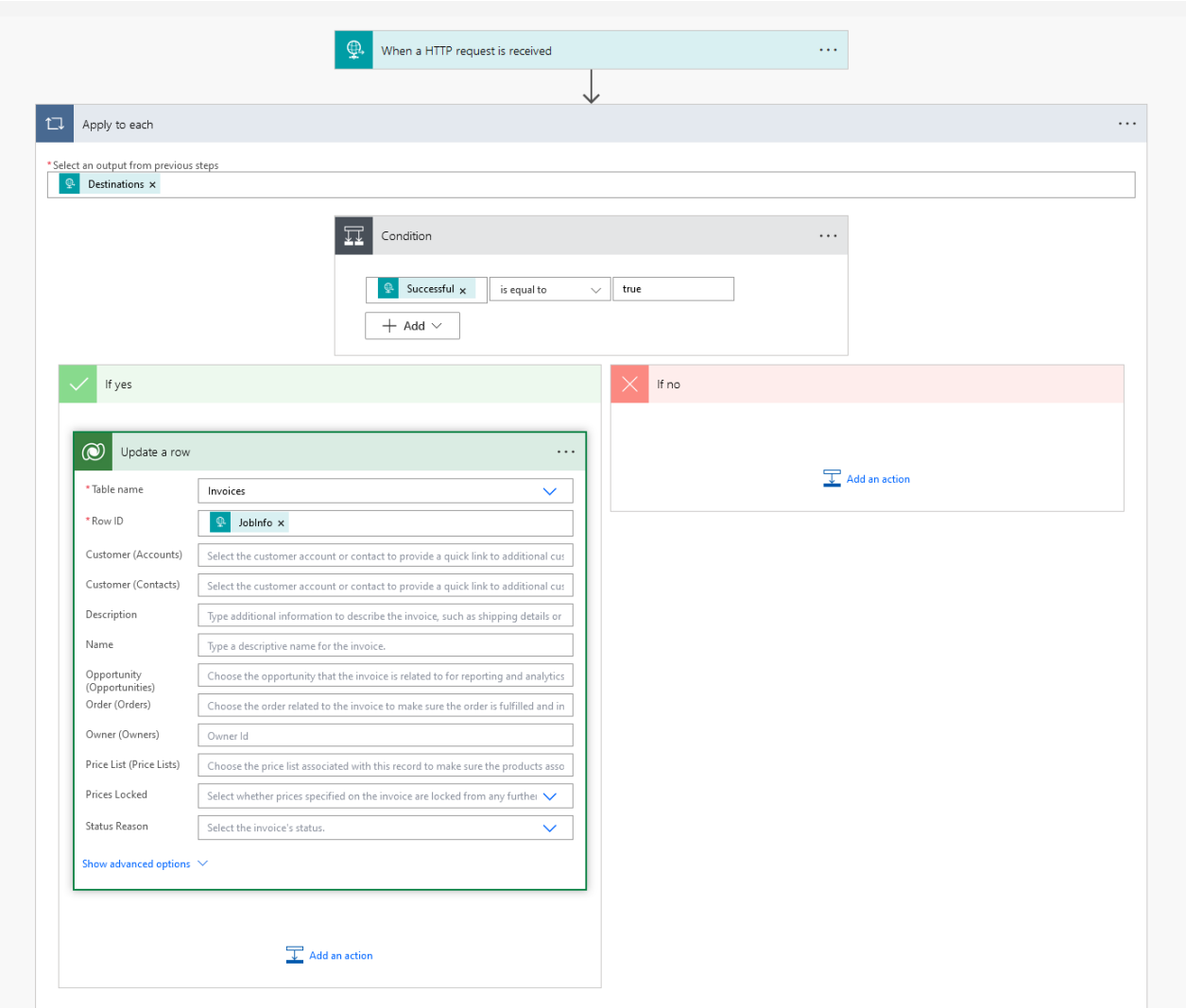
Ein HTTP-POST-Request Flow kann aus einem Beispiel-JSON ein passendes Schema generieren. Dies ist notwendig, damit man im Flow auf die Werte im JSON zugreifen kann. Eine Beispielstatusmeldung findet man unter `https://{host}:3216/webapi/v2/doc/index.html#/default/post_webapi_v2_status`, im Textfeld “Example Value”. Diese kopiert und fügt man in den Callback Flow ein, und das Schema wird generiert.




Invoice EPost Callback
Undo
Redo
Comments
Save
Flow checker
Test
When a HTTP request is received
*Who can trigger the flow?
Anyone
HTTP POST URL
https://
Request Body JSON Schema
Enter or paste a sample JSON payload.
"Base64Content": "string"
}
],
"JobInfo": "string",
"StatusUrls": [
"string"
],
"Direction": "send"
}
Done


Invoice EPost Callback
Undo
Redo
Comments
Save
Flow checker
Test
When a HTTP request is received
*Who can trigger the flow?
Anyone
Copy Url
HTTP POST URL
https://
Request Body JSON Schema
{
"type": "object",
"properties": {
"Destinations": {
"type": "array",
"items": {
"type": "object",
"properties": {
"Destination": {
Use sample payload to generate schema
Show advanced options

Danach kann man auf Elemente der *JobInfo*-Struktur im Post-Requests zugreifen, um z.B. die ID zu nutzen um den Datenbank-Record wiederzufinden und entsprechend zu aktualisieren:



4.3. Send-Flow

Der Send-Flow reagiert auf z.B. Schalter “Rechnung als Brief senden” (Datenbank basiert) und sendet einen HTTP-Request an die Web-API mit den Daten im JSON-Format inklusive der URL des Callback-Flows für die Rückmeldung des Status.

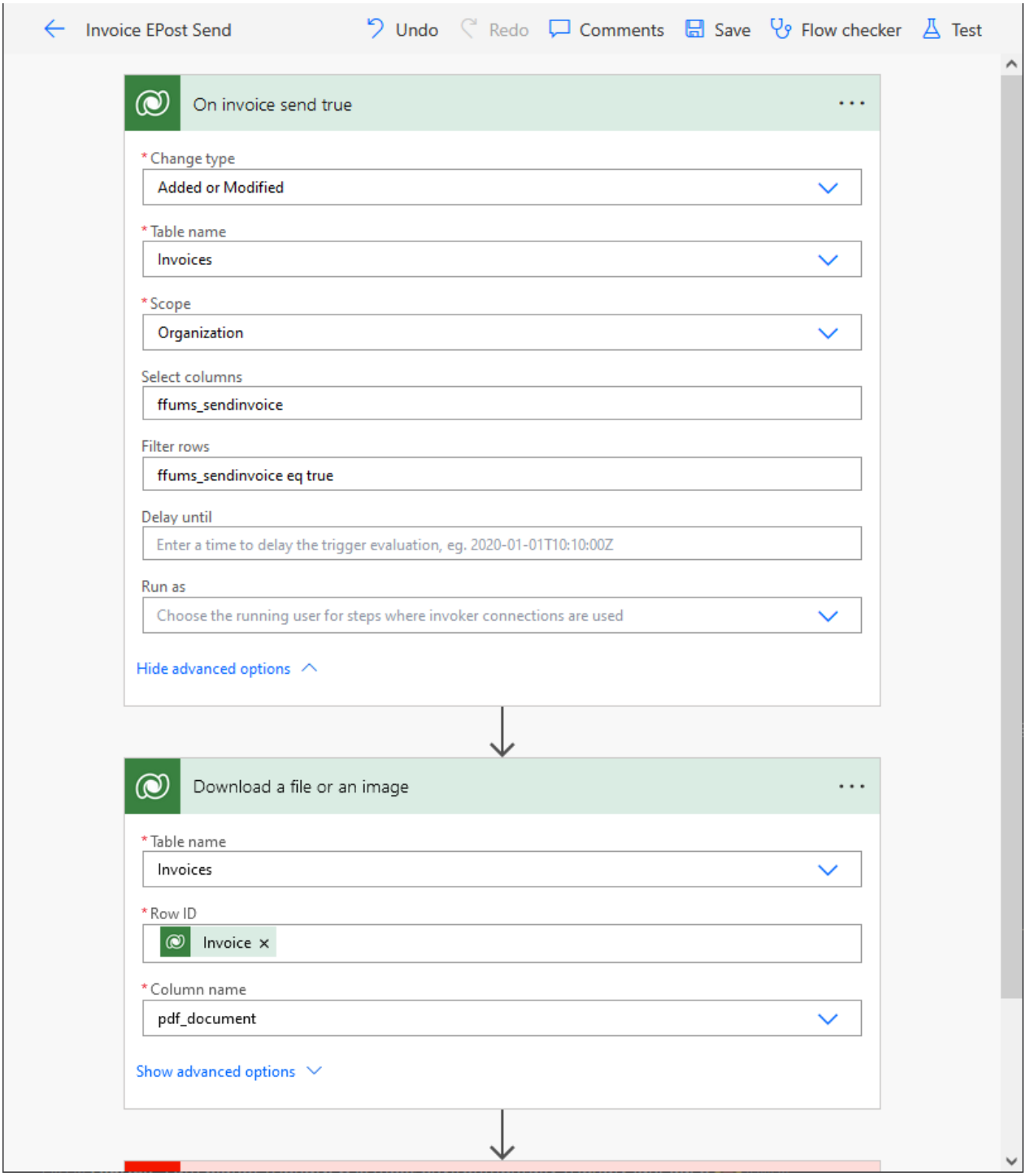
Im HTTP-Request wird eine *Transfer Request*-Struktur übergeben, welche Informationen zum Sendeauftrag enthält. Für den Einsatz mit Dynamics 365 enthält diese z.B. folgende Felder:

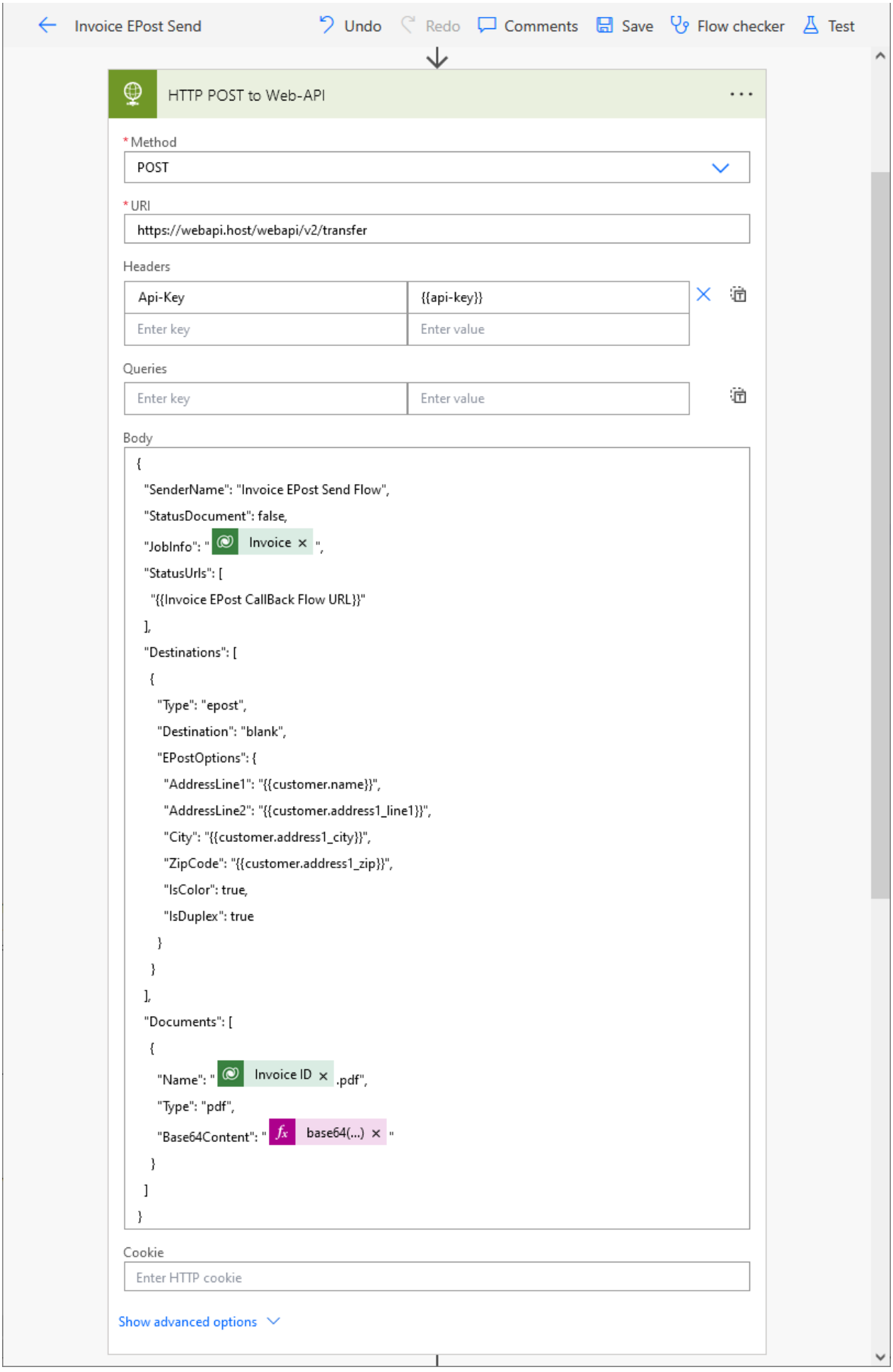
```
{
  "SenderName": "Invoice EPost Send Flow",
  "StatusDocument": false,
  "JobInfo": "@{triggerOutputs()?['body/invoiceid']}",
  "StatusUrls": [
    "{{Invoice EPost CallBack Flow URL}}"
  ],
  "Destinations": [
    {
      "Type": "epost",
      "Destination": "blank",
      "EPostOptions": {
        "AddressLine1": "{{customer.name}}",
        "AddressLine2": "{{customer.address1_line1}}",
        "City": "{{customer.address1_city}}",
        "ZipCode": "{{customer.address1_zip}}",
        "IsColor": true,
        "IsDuplex": true
      }
    }
  ],
  "Documents": [
    {
      "Name": "@{triggerOutputs()?['body/invoicenumber']}.pdf",
      "Type": "pdf",
      "Base64Content": "@{base64(outputs('Get_invoice_pdf')?['body'])}"
    }
  ]
}
```

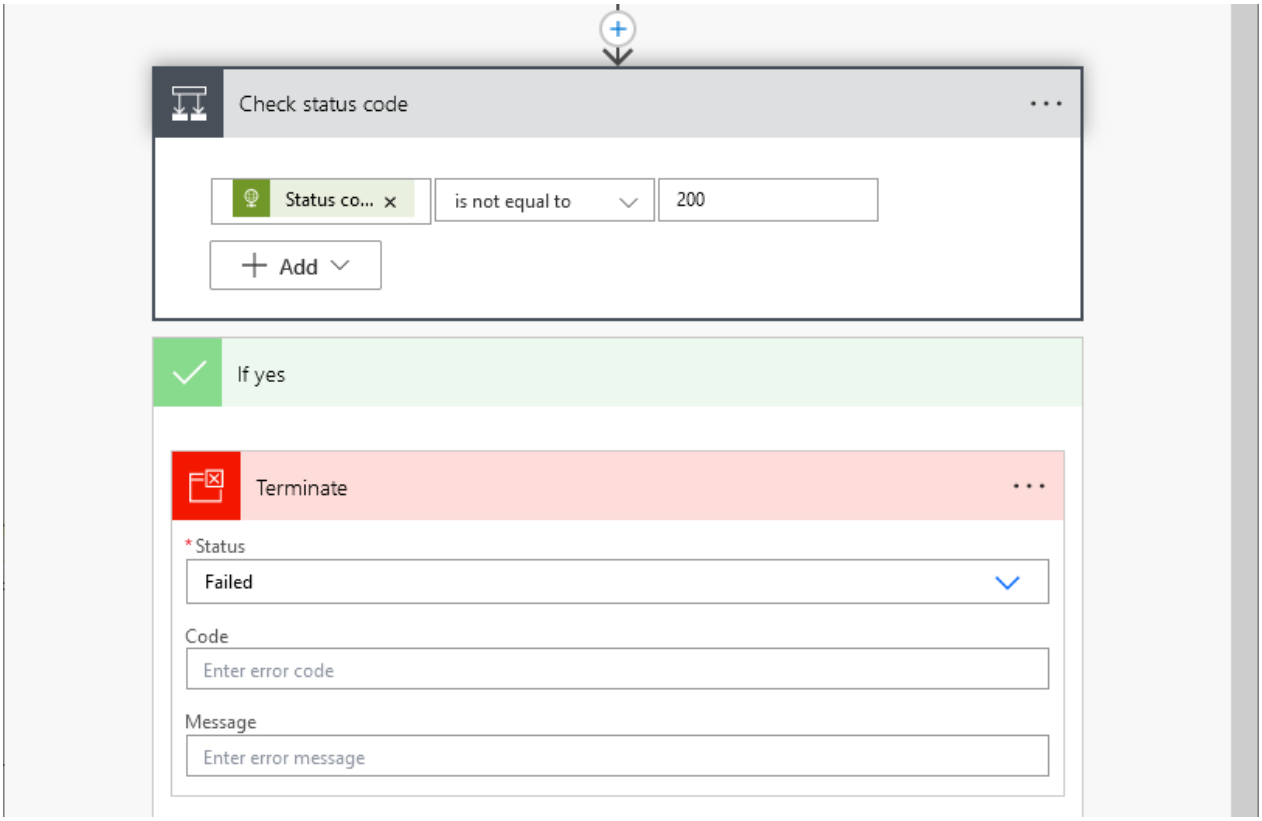
Die ID der Rechnung sollte in der *JobInfo*-Struktur übertragen werden, damit diese im Callback Flow zugeordnet werden kann. Das PDF muss Base64 kodiert übertragen werden.

Der API-Key muss hier im Header an die WebAPI-Komponente übertragen werden.

Der Statuscode des Requests kann auf Fehler überprüft und entsprechend behandelt werden, z.B. durch zurücksetzen des Schalters und eine Fehlermeldung.




Invoice EPost Send
Undo
Redo
Comments
Save
Flow checker
Test
HTTP POST to Web-API
*Method
POST
*URI
https://webapi.host/webapi/v2/transfer
Headers
Api-Key
{{api-key}}
Enter key
Enter value
Queries
Enter key
Enter value
Body
{
"SenderName": "Invoice EPost Send Flow",
"StatusDocument": false,
"JobInfo": "
Invoice
",
"StatusUrls": [
"{{Invoice EPost CallBack Flow URL}}"
],
"Destinations": [
{
"Type": "epost",
"Destination": "blank",
"EPostOptions": {
"AddressLine1": "{{customer.name}}",
"AddressLine2": "{{customer.address1_line1}}",
"City": "{{customer.address1_city}}",
"ZipCode": "{{customer.address1_zip}}",
"IsColor": true,
"IsDuplex": true
}
}
],
"Documents": [
{
"Name": "
Invoice ID
.pdf",
"Type": "pdf",
"Base64Content": "
base64(...)
"
}
]
}
Cookie
Enter HTTP cookie
Show advanced options



4.3.1. Weitere Informationen

Eventuell ist es notwendig, neue Attribute in der Datenbank anzulegen, z.B. einen “Rechnung als Brief senden”-Schalter, ein “Rechnungs-PDF”-Attribut, oder für den Auslandsbriefversand ein Attribut “Ländername” auf Deutsch in Großbuchstaben (z.B. ÖSTERREICH).

Dokumentation für die JSON-Struktur findet man unter `https://{host}:3216/webapi/v2/doc/`.

Die *JobInfo*-Struktur kann beliebig angepasst werden und wird genauso wieder in der Statusmeldung zurückgegeben, wie sie übertragen wurde. Das Schema im Callback Flow muss angepasst werden, falls man die *JobInfo*-Struktur ändert.

Prinzipiell ist es möglich, die WebAPI-Komponente mit jedem CRM oder ERP einzusetzen, welches HTTP-Requests senden und empfangen kann.

4.4. Nutzungsbeispiel: Ferrari electronic AG

Die Ferrari electronic AG nutzt ein Dynamics 365 CRM zur Unterstützung der Vertriebsprozesse. Dabei wurden Flows eingerichtet, welche dazu dienen, Dokumente und Nachrichten aus dem CRM als

- Fax,
- EPost-Brief,
- SMS und
- Druckaufträge

an die OfficeMaster Suite zu übertragen. Nach Abschluss der Übertragung erfolgt eine Rückmeldung an das CRM und der Vorgang wird in der Datenbank als abgeschlossen markiert.

Wenn in den Stammdaten der Kunden gepflegt wird, welches die präferierte Kommunikationsform ist, kann dann automatisch der entsprechende Versandweg genutzt werden (E-Mail, Brief oder Fax).

5. Go Code-Beispiel zur Nutzung der Web-API

5.1. Hinweis

Code-Beispiele dienen der Unterstützung des Entwicklungsprozesses beim Kunden. Der Code in diesen Beispielen unterliegt einigen Einschränkungen:

- Ferrari electronic AG bietet keinen Support für Code-Beispiele.
- Der Einsatz in Produktivumgebungen ohne vorherige Maßnahmen zur Qualitätssicherung ist nicht empfohlen.

5.2. Go Code

Beispielimplementation Dokumentenversand via Fax (Go):

```
func main() {
    // Start webserver to receive status messages
    server := startStatusServer()

    filename := "letter.pdf"
    base64Data, err := readFileAsBase64(filename)
    if err != nil {
        fmt.Println(err)
        return
    }

    // Using map[string]interface{} for simplicity, creating dedicated
    // structs would be desired
    jsonMap := map[string]interface{}{
        "SenderName":     "Golang WebAPI Example",
        "StatusDocument": false, // If true, the document base64 will b
                                 // be provided in the status response
        "JobInfo": map[string]interface{}{
            // Anything entered into JobInfo will be copied to the response,
            // useful for providing identifiers
            "uuid": "08155926-61b9-4424-9958-5ec25624a620",
        },
        // This is the url of our status server
        "StatusUrls": []string{"http://localhost:48000/status"},
        // see openapi documentation at
        // https://localhost:3216/webapi/v2/doc/index.html
        "Destinations": []map[string]interface{}{
            {
                "Type":        "fax",
                "Destination": "+49 0123 4567890",
                "FaxOptions": map[string]interface{}{
                    "Header":             "HEADER",
                    "Csid":               "+49 0123 1234567",
                    "CallingPartyNumber": "+49 0123 1234567",
                },
            },
        },
```

```
    // It is not recommended to provide more than one document
    //  per request
    "Documents": []map[string]interface{}{
        {
            "Name":          filename,
            "Type":          "pdf",
            "Base64Content": base64Data,
        },
    },
}

//Encode data structure to json bytes
jsonData, err := json.Marshal(jsonMap)
if err != nil {
    fmt.Println(err)
    return
}

//Write json bytes to a reader
reader := bytes.NewReader(jsonData)

//Create request to web-api
req, err := http.NewRequest(http.MethodPost,
    "https://localhost:3216/webapi/v2/transfer", reader)
if err != nil {
    fmt.Println(err)
    return
}
req.Header.Set("api-key", "<api-key>")
req.Header.Set("Content-Type", "application/json")

// Skip certificate verification, it may be rejected
// because it is self-signed (this is insecure - for testing)
http.DefaultTransport.(*http.Transport).TLSClientConfig =
    &tls.Config{InsecureSkipVerify: true}

//Create http client to handle requests
client := http.Client{}
//Submit POST Request to Web-API
resp, err := client.Do(req)
//Catch system errors
if err != nil {
    fmt.Println(err)
```

```
        return
    }

    //Check response for success
    if resp.StatusCode != http.StatusOK {
        fmt.Printf("Received non-normal status code: %d\n", resp.StatusCode)
        respData, err := io.ReadAll(resp.Body)
        if err != nil {
            fmt.Println(err)
        } else {
            fmt.Println(string(respData))
        }
        return
    }

    // Wait 3 Minutes to receive status message, server messages
    // are still being received (transmission may take even longer than 3 mins)
    // it would be better to wait for the response code here
    time.Sleep(time.Minute * 3)

    //Shutdown status server after 3 minutes and exit program
    server.Shutdown(context.Background())
}

func readFileAsBase64(filename string) (string, error) {
    data, err := os.ReadFile(filename)
    if err != nil {
        return "", err
    }

    return base64.StdEncoding.EncodeToString(data), nil
}

// Starts a webserver on port 48000 and prints incoming messages.
// Used for receiving status messages from jobs
func startStatusServer() *http.Server {
    mux := http.NewServeMux()
    mux.HandleFunc("/status", func(w http.ResponseWriter, r *http.Request) {
        body, err := io.ReadAll(r.Body)
        if err != nil {
            fmt.Println(err)
            return
        }
```

```
        fmt.Println(string(body))

        w.WriteHeader(http.StatusOK)
    })
    server := http.Server{
        Addr:    ":48000",
        Handler: mux,
    }
    go server.ListenAndServe()

    //Wait for webserver to fully start
    time.Sleep(time.Millisecond * 100)

    return &server
}
```

Für den Versand von EPost-Briefen kann folgende `jsonMap` mit obigem Beispiel-Code verwendet werden:

```
jsonMap := map[string]interface{}{
    "SenderName":     "Golang WebAPI Example",
    "StatusDocument": false, // If true, the document base64 will be
                             // provided in the status response
    "JobInfo": map[string]interface{}{
        // Anything entered into JobInfo will be copied to the response,
        // useful for identifying the response
        "uuid": "8ba612b5-2c07-45fb-9c13-3a0af47e5a76",
    },
    // This is the url of our status server
    "StatusUrls": []string{"http://localhost:48000/status"},
    // see openapi documentation at
    // https://localhost:3216/webapi/v2/doc/index.html
    "Destinations": []map[string]interface{}{
        "Type":        "epost",
        // Destination is ignored for EPost,
        // but value needs to be set and non-empty
        "Destination": "thisisignored",
        // Please follow required fields of the official EPost API:
        // https://api.epost.docuguide.com/swagger/index.html#/
        "EPostOptions": map[string]interface{}{
            {
            "AddressLine1": "Example Company Ltd.",
```

```
            "AddressLine2": "Example Street 123",
            "ZipCode": "12345",
            "City": "Exampletown",
            "Country": "COUNTRY",
            },
        },
    },
    // It is not recommended to provide more than one document
    // per request, merge multiple documents into one on client side
    "Documents": []map[string]interface{}{
        {
            "Name":          filename,
            "Type":          "pdf",
            "Base64Content": base64Data,
        },
    },
}
```