CRM Export – Salesforce Lead Transfer

Vollständige Dokumentation des Feldmapping-Prozesses

Version: 1.0 **Datum:** 10.02.2026 **Projekt:** LSPortal – Salesforce Integration

---

1. Übersicht

Der CRM Export ermöglicht es, Kontaktdaten aus dem LSPortal als Leads an Salesforce zu übertragen. Der Benutzer kann über den **Field Configurator** festlegen, welche Felder übertragen werden und wie sie in Salesforce benannt sein sollen.

Beteiligte Komponenten

Komponente	Beschreibung
Field Configurator (CRM Settings)	UI zur Konfiguration der Felder
LS_FieldMappings (DB View)	Speicherung der Konfiguration (ConfigData als JSON)
LS_LeadReport (DB View)	Kontaktdaten des Events
salesforceLeadLib.js	Hauptbibliothek für Transfer-Logik
Salesforce Backend	Node.js-Server (OAuth + API-Weiterleitung)
Salesforce Lead API	Ziel-API zum Erstellen von Leads

---

2. Konfiguration (Field Configurator)

2.1 Felder laden

Beim Öffnen des Field Configurators werden alle verfügbaren Felder aus mehreren Quellen geladen:

1. **API-Felder**: Aus `LS_LeadReport` (alle Felder des Events)
2. **Fallback**: Standard-SF-Lead-Felder (22 Felder wie LastName, Company, Email, etc.)
3. **Gespeicherte Konfiguration**: Aus `LS_FieldMappings.ConfigData` (DB)

Die Ladereihenfolge ist optimiert:

- Schritt 1: Sofortige Anzeige mit Standard-Feldern (kein API-Aufruf)
- Schritt 2: API-Felder im Hintergrund laden und Grid aktualisieren
- Schritt 3: Gespeicherte Konfiguration aus DB anwenden

2.2 Felder konfigurieren

Der Benutzer kann folgende Aktionen durchführen:

Aktion	Beschreibung
Aktivieren/Deaktivieren	Checkbox pro Feld – nur aktive Felder werden übertragen
Custom Label setzen (✏️)	SF-Feldname ändern – wird im Payload als Feldname verwendet
Custom Field hinzufügen (+)	Neues Feld mit SF-Feldname und optionalem Standardwert
Custom Field löschen (🗑️)	Benutzerdefiniertes Feld entfernen
Filtern	Nach Kategorie: Alle, Aktiv, Inaktiv, Pflicht, Custom

Suchen	Felder nach Name durchsuchen

Pflichtfelder: `LastName` und `Company` sind immer aktiv und können nicht deaktiviert werden (Salesforce-Anforderung).

2.3 Custom Labels (Feldmapping)

Das Custom Label ist **kein reines Anzeigelabel**, sondern ein echtes **Mapping auf den SF-Feldnamen**.

Beispiel:

LS-Feld	Custom Label	SF Payload
`Answer01`	`Branche`	`Branche__c`
`KontaktViewId`	`ContactViewId`	`ContactViewId__c`
`Question01`	`Reward`	`Reward__c`
`Question01`	*(kein Label)*	`Question01__c`
`LastName`	*(egal)*	`LastName` (Standard-Feld)

Wichtig: Das gemappte Feld muss in Salesforce existieren. Andernfalls gibt Salesforce eine Fehlermeldung zurück.

2.4 Speicherung

Die gesamte Konfiguration wird als JSON in `LS_FieldMappings.ConfigData` gespeichert:

```
{
  "fieldConfig": {
    "config": {
      "fields": [
        { "fieldName": "LastName", "active": true },
        { "fieldName": "Answer01", "active": true },
        { "fieldName": "KontaktViewId", "active": false }
      ]
    }
  },
  "customLabels": {
    "KontaktViewId": "ContactViewId",
    "Answer01": "Branche",
    "Question01": "Reward"
  },
  "customFields": [
    {
      "id": "cf_1234",
      "sfFieldName": "Area__c",
      "value": "Germany",
      "active": true
    }
  ]
}
```

---

3. Transfer-Prozess

3.1 Fall 1: Event MIT Kontakten

Funktion: `openCrmExport(rootElement, contactId)`

```
1. Kontaktdaten von LS_LeadReportById laden
   → Echte Werte (FirstName: "Gilbert", LastName: "Schwaab", ...)

2. Feldkonfiguration von LS_FieldMappings laden
   → fieldConfig, customLabels, customFields

3. CustomLabels in die SalesforceLeadLib-Instanz synchronisieren
   → instance.fieldMappingService.customLabels = customLabels

4. Aktive Felder mit echten Werten in der UI anzeigen
   → Tabellen- oder Kartenansicht

5. Benutzer prüft die Daten und klickt "Transfer to Salesforce"

6. Payload wird erstellt und an das Backend gesendet
```

3.2 Fall 2: Event OHNE Kontakte (Test-Modus)

Funktion: `openCrmTestExport(rootElement, eventId)`

```
1. Keine Kontaktdaten vorhanden
   → Standard-Feldstruktur wird verwendet (26 Felder)

2. Feldkonfiguration von LS_FieldMappings laden
   → fieldConfig, customLabels, customFields

3. CustomLabels in die SalesforceLeadLib-Instanz synchronisieren
   → instance.fieldMappingService.customLabels = customLabels

4. fakeDataGenerator erzeugt realistische Testdaten
   → FirstName: "Max", LastName: "Müller", Company: "Tech GmbH", ...

5. Benutzer kann alle Testwerte bearbeiten
   → Änderungen werden in localStorage gespeichert

6. Benutzer klickt "Transfer to Salesforce"

7. Payload wird erstellt (gleiche Logik wie Fall 1)
```

---

4. Payload-Erstellung (Detaillogik)

4.1 Entscheidungsbaum

Für jedes aktive Feld wird der SF-Feldname wie folgt bestimmt:

```
Ist das Feld ein Standard-SF-Feld?
(LastName, Company, Email, Phone, FirstName, ...)
│
├── JA → Feldname bleibt unverändert
│        Beispiel: "LastName" → "LastName"
│
└── NEIN → Hat das Feld ein Custom Label?
           │
           ├── JA → Custom Label als Feldname verwenden
```

```
│        Beispiel: "Answer01" + Label "Branche" → "Branche"
│
└── NEIN → Original-LS-Feldname verwenden
          Beispiel: "Answer01" → "Answer01"
│
└── Endet der Name mit __c?
    │
    ├── JA → Nichts tun
    │        Beispiel: "Area__c" → "Area__c"
    │
    └── NEIN → __c anhängen
               Beispiel: "Branche" → "Branche__c"
```

4.2 Standard-SF-Lead-Felder (kein __c)

Diese 22 Felder werden ohne `__c` -Suffix gesendet:

Feld	Feld	Feld
LastName	FirstName	Company
Email	Phone	MobilePhone
Title	Website	Street
City	State	PostalCode
Country	Description	Industry
AnnualRevenue	NumberOfEmployees	LeadSource
Status	Rating	Salutation
Fax		

4.3 Numerische Felder

Die Felder `AnnualRevenue` und `NumberOfEmployees` werden automatisch in Zahlen konvertiert. Salesforce erwartet numerische Werte, keine Strings. Ungültige Werte werden aus dem Payload entfernt.

4.4 Beispiel-Payload

```
{
  "leadData": {
    "FirstName": "Gilbert",
    "LastName": "Schwaab",
    "Company": "Convey GmbH",
    "Email": "schwaab@convey.de",
    "Phone": "+49 160 926 78 073",
    "Branche__c": "IT",
    "Reward__c": "Gold Status",
    "Area__c": "Germany"
  },
  "attachments": [],
  "leadId": "64331eac-..."
}
```

---

5. Backend → Salesforce

5.1 Endpunkt

```
POST {backendUrl}/api/salesforce/leads
Headers: Content-Type: application/json, X-Org-Id: {orgId}
Credentials: include (Session-Cookie)
```

5.2 Authentifizierung

Das Backend verwendet **OAuth 2.0 Authorization Code Flow**:

1. Benutzer klickt "Connect to Salesforce"
2. OAuth-Popup öffnet sich → Salesforce-Login
3. Callback mit Authorization Code → Backend tauscht gegen Access Token
4. Token wird in der Server-Session gespeichert
5. Alle API-Aufrufe verwenden den gespeicherten Token

5.3 Antworten

Status	Bedeutung	Beispiel
200	Erfolg	Lead erstellt, SF-Lead-ID zurückgegeben
400	Validierungsfehler	Fehlende Pflichtfelder, ungültige Feldnamen
409	Duplikat	Lead existiert bereits in SF
500	Serverfehler	Backend- oder SF-API-Fehler

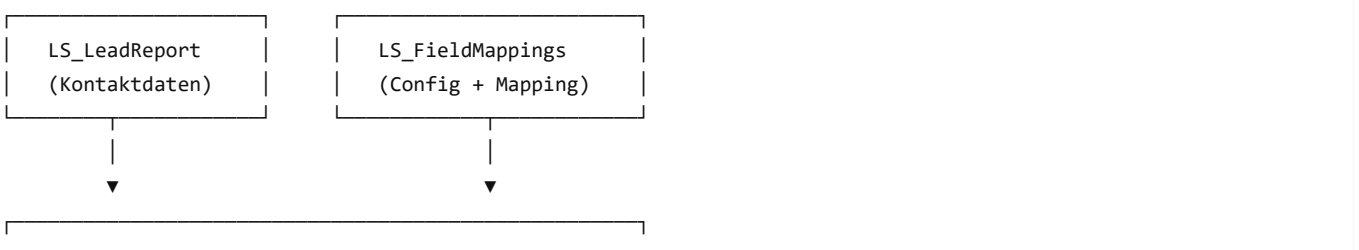
5.4 Fehlermeldung bei fehlenden Feldern

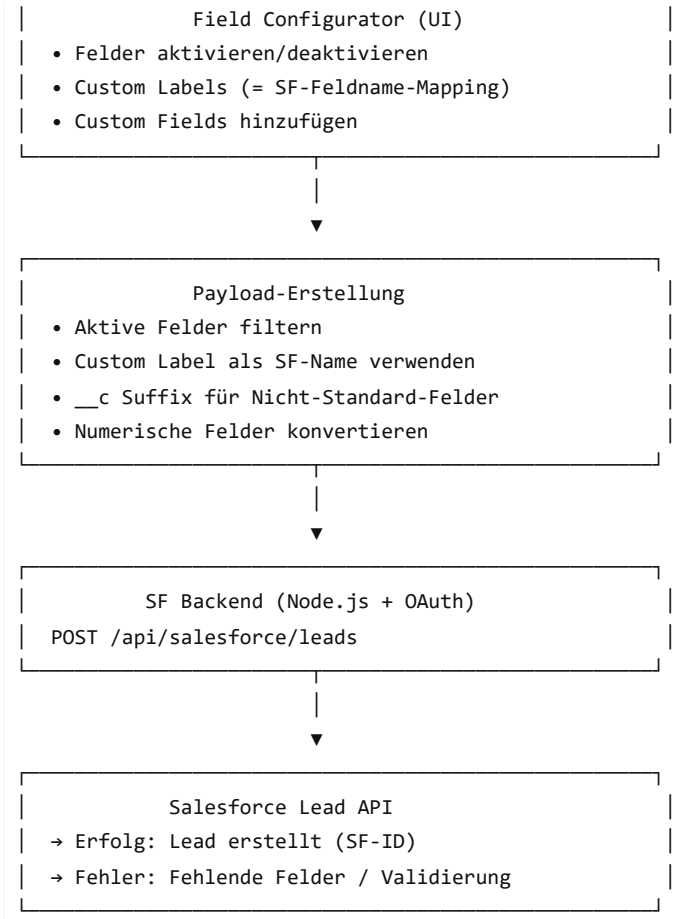
Wenn Felder im Payload nicht in Salesforce existieren, zeigt das System eine detaillierte Fehlermeldung an:

```
Missing Fields in Salesforce

The following fields do not exist in Salesforce:
ContactViewId__c, Suffix__c, MiddleName__c

Please ensure field names match exactly as they appear in Salesforce.

Important notes:
• Field names are case-sensitive
• Custom fields must end with __c
• Spaces in field names are not allowed
• Standard field names must match exactly
```

---

6. Datenfluss (Zusammenfassung)





---

7. Fehlerbehebungen (10.02.2026)

Commit	Beschreibung
`19d63f01`	Custom Labels als reines Display-Label behandelt (später korrigiert)
`b5c9a3b9`	Unbenutzte `getResourceText`-Funktion entfernt, ~100 redundante Kommentare bereinigt
`fab1e011`	**CustomLabels-Sync**: CustomLabels aus der DB werden jetzt korrekt in die Instanz synchronisiert, damit der Transfer den gemappten SF-Feldnamen verwendet

Kernfix: CustomLabels Synchronisation

Problem: CustomLabels wurden aus `LS_FieldMappings.ConfigData` geladen, aber nicht in `instance.fieldMappingService.customLabels` synchronisiert. Der Transfer-Code griff auf die leere Instanz-Variable zu und verwendete daher immer den Original-Feldnamen.

Lösung: Nach dem Laden der ConfigData werden die CustomLabels explizit in die Instanz synchronisiert:

```
const instance = this._getInstance();
if (instance) {
    instance.fieldMappingService.customLabels = customLabels;
}
```

Dies wurde in beiden Transfer-Pfaden implementiert:

- `openCrmExport` (realer Transfer mit Kontaktdaten)
- `openCrmTestExport` (Test-Transfer mit generierten Daten)