

# 1Password CLI: JSON Read/Edit Workflow for LOGIN Items

---

**Purpose:** Document for 1Password Support troubleshooting

**Date:** 2025-12-07

**CLI Version:** 1Password CLI v2

---

## Problem Summary

We need to programmatically add REFERENCE fields (Related Items links) to existing LOGIN items. The field assignment syntax (`op item edit <uuid> "field=value"`) does not support creating REFERENCE type fields, so we must use the JSON stdin workflow.

---

## Workflow Overview

- |   |
|---|
| 1. READ: <code>op item get &lt;uuid&gt; --format json</code><br>→ Returns complete item JSON  |
| 2. MODIFY: Parse JSON, add REFERENCE field to fields array<br>→ Ensure proper section structure<br>→ Remove problematic fields ( <code>category_id</code> if present) |
| 3. WRITE: <code>echo '&lt;json&gt;'   op item edit &lt;uuid&gt; -</code><br>→ Pipe modified JSON back via stdin   |

---

## Step 1: Read Item

```
op item get <uuid> --format json
```

### Sample Response (LOGIN item)

```
{
  "id": "abc123...",
  "title": "PayPal",
  "version": 5,
  "vault": {
    "id": "vault-uuid",
    "name": "Private"
  },
}
```

```
"category": "LOGIN",
"urls": [
  {
    "label": "website",
    "primary": true,
    "href": "https://www.paypal.com/signin"
  }
],
"sections": [
  {
    "id": "section1",
    "label": "Custom Section"
  }
],
"fields": [
  {
    "id": "username",
    "type": "STRING",
    "purpose": "USERNAME",
    "label": "username",
    "value": "user@example.com"
  },
  {
    "id": "password",
    "type": "CONCEALED",
    "purpose": "PASSWORD",
    "label": "password",
    "value": "REDACTED"
  },
  {
    "id": "field123",
    "section": {
      "id": "section1",
      "label": "Custom Section"
    },
    "type": "STRING",
    "label": "Custom Field",
    "value": "custom value"
  }
],
"createdAt": "2012-04-27T19:08:14Z",
"updatedAt": "2025-12-03T14:22:21Z"
}
```

**Key observations:**

- **category** is "LOGIN" (uppercase in CLI v2 output)
- Fields array contains all fields with their **type**, **label**, **value**
- Section membership is indicated by **section** object within each field
- Fields without **section** are in the default/top-level section

## Step 2: Modify JSON

### Adding a REFERENCE Field

To create a "Related Items" link, we add a field with `type: "REFERENCE"`:

```
import json
import uuid

# Parse the item
item_data = json.loads(op_output)

# Ensure sections array exists
if "sections" not in item_data:
    item_data["sections"] = []

# Find or create "Related Items" section
section_id = None
for section in item_data["sections"]:
    if section.get("label") == "Related Items":
        section_id = section.get("id")
        break

if not section_id:
    section_id = "related_items"
    item_data["sections"].append({
        "id": section_id,
        "label": "Related Items"
    })

# Ensure fields array exists
if "fields" not in item_data:
    item_data["fields"] = []

# Add the REFERENCE field
new_field = {
    "id": str(uuid.uuid4()).replace("-", "")[:26], # Generate unique ID
    "section": {
        "id": section_id,
        "label": "Related Items"
    },
    "type": "REFERENCE",
    "value": "target-item-uuid-here" # UUID of item to link to
}

item_data["fields"].append(new_field)
```

### Critical: Remove `category_id` if Present

Some items (especially older or CUSTOM category items) have a `category_id` field that causes "identity inconsistency" errors during edit:

```
# MUST remove category_id before editing
if "category_id" in item_data:
    del item_data["category_id"]
```

**Error if not removed:**

```
[ERROR] 2025/12/07 10:30:45 item identity inconsistency
```

---

## Step 3: Write Item Back

```
echo '<modified-json>' | op item edit <uuid> -
```

### Python Implementation

```
import subprocess
import json

result = subprocess.run(
    ["op", "item", "edit", item_uuid, "-"],
    input=json.dumps(item_data),
    capture_output=True,
    text=True,
    check=True,
    timeout=30,
)
```

The `-` argument tells `op` to read the item JSON from stdin.

---

## Complete Working Example

```
import subprocess
import json
import uuid

def add_related_item_link(source_uuid: str, target_uuid: str) -> bool:
    """Add a Related Items link from source to target item."""

    # Step 1: Read current item
    result = subprocess.run(
        ["op", "item", "get", source_uuid, "--format", "json"],
        capture_output=True,
```

```
        text=True,
        check=True,
        timeout=30,
    )
    item_data = json.loads(result.stdout)

    # Step 2: Prepare sections
    if "sections" not in item_data or item_data["sections"] is None:
        item_data["sections"] = []
    if "fields" not in item_data or item_data["fields"] is None:
        item_data["fields"] = []

    # Find or create Related Items section
    section_id = None
    for section in item_data["sections"]:
        if section.get("label") == "Related Items":
            section_id = section.get("id")
            break

    if not section_id:
        section_id = "related_items"
        item_data["sections"].append({
            "id": section_id,
            "label": "Related Items"
        })

    # Check if link already exists
    for field in item_data["fields"]:
        if field.get("type") == "REFERENCE" and field.get("value") ==
target_uuid:
        print("Link already exists")
        return True

    # Add REFERENCE field
    new_field = {
        "id": str(uuid.uuid4()).replace("-", "")[:26],
        "section": {
            "id": section_id,
            "label": "Related Items"
        },
        "type": "REFERENCE",
        "value": target_uuid
    }
    item_data["fields"].append(new_field)

    # CRITICAL: Remove category_id to avoid identity inconsistency error
    if "category_id" in item_data:
        del item_data["category_id"]

    # Step 3: Write back
    result = subprocess.run(
        ["op", "item", "edit", source_uuid, "-"],
        input=json.dumps(item_data),
        capture_output=True,
```

```
        text=True,  
        check=True,  
        timeout=30,  
    )  
  
    return True
```

---

## Known Issues & Error Messages

### 1. "a field to edit must have either an ID or a Label"

**Cause:** Malformed field in the JSON (missing `id` or `label`)

**Solution:** Clean up fields before submitting:

```
cleaned_fields = []  
for field in item_data["fields"]:  
    if field.get("id") or field.get("label"):  
        cleaned_fields.append(field)  
item_data["fields"] = cleaned_fields
```

### 2. "item identity inconsistency"

**Cause:** `category_id` field present in item JSON

**Solution:** Remove before editing:

```
if "category_id" in item_data:  
    del item_data["category_id"]
```

### 3. CUSTOM Category Items Cannot Be Edited

**Cause:** Items with `category: "CUSTOM"` AND `category_id` field fail JSON round-trip

**Solution:** Convert to `SECURE_NOTE` first, or use a different approach

### 4. Cross-Vault References Not Supported

**Cause:** 1Password does not support REFERENCE fields linking items in different vaults

**Solution:** Ensure both items are in the same vault before linking

---

## Field Types Reference

Type	Description	Example Value
------	-------------	---------------

Type	Description	Example Value
STRING	Plain text	"hello"
CONCEALED	Hidden/password	"secret123"
URL	Web URL	"https://example.com"
EMAIL	Email address	"user@example.com"
DATE	Unix timestamp	1733529600
REFERENCE	Link to another item	"item-uuid"
FILE	File attachment	(special handling)

## Troubleshooting Commands

```
# Check if CLI is authenticated
op account get --format json

# Get item in human-readable format (for debugging)
op item get <uuid>

# Get item in JSON format (for programmatic use)
op item get <uuid> --format json

# Check item category and structure
op item get <uuid> --format json | jq '.category, .fields | length'

# List Related Items fields
op item get <uuid> --format json | jq '.fields[] | select(.type == "REFERENCE")'
```

## Questions for 1Password Support

1. Is the JSON stdin workflow (`op item edit <uuid> -`) the recommended way to add REFERENCE fields?
2. Why does `category_id` cause "identity inconsistency" errors? Should we always strip it before editing?
3. For CUSTOM category items with `category_id`, is there a way to edit them via CLI, or must they be converted to another category first?
4. Is there documentation on which fields are required vs optional when submitting JSON via stdin?
5. Are there any differences in the JSON schema between item categories (LOGIN vs SECURE\_NOTE vs CUSTOM) that we should be aware of?