

Testen von Use Cases

Vertragsmanagement

TC 5.1: Contract Synchronisation (Einzel)

Ziel: Vertrag wird aus Gridware nach ERPNext synchronisiert.

Testschritte:

1. Gehe zu becharged Settings → Feld "Gridware Contract ID"
2. Gib eine Contract ID ein (z.B. "12345")
3. Klicke Button "Fetch Contract by ID"
4. **Validieren:**
 - Gridware Contract wurde erstellt (Name: "GWC-.....")
 - Alle Felder sind gefüllt: contract_id, contract_number, contract_type
 - active_from / active_until sind gesetzt
 - VAT ist korrekt übertragen

Erfolgskriterium: Contract ist bereit für Ladevorgang-Verarbeitung.

TC 5.2: Contract Customers Sync

Ziel: Kunden-Liste des Contracts wird synchronisiert.

Testschritte:

1. Öffne den Contract aus TC 5.1
2. Scrolla zu "Contract Customers" Tab
3. **Validieren:**
 - Mindestens eine Zeile vorhanden
 - customer_id, subject_id, subject_name sind gefüllt
 - `gridware_party` Link ist gesetzt (Background Job)
 - `customer` Link ist gesetzt zu ERPNext Customer

Erfolgskriterium: Alle Kunden sind korrekt verlinkt.

TC 5.3: Contract Supplier Sync

Ziel: Lieferant wird korrekt verlinkt.

Testschritte:

1. Öffne den Contract aus TC 5.1
2. Gehe zu "Supplier Details" Section
3. **Validieren:**
 - `gridware_supplier_party` ist gesetzt
 - `supplier` Link zeigt zu ERPNext Supplier
 - Supplier Name ist korrekt

Erfolgskriterium: Supplier kann in Lieferantenabrechnung verwendet werden.

TC 5.4: Contract Usage Fetch

Ziel: Ladevorgänge für Contract werden abgerufen.

Testschritte:

1. Öffne Contract aus TC 5.1
2. Schaue in Links: "Usage" Sektion
3. **Validieren:**
 - Mindestens ein Gridware Usage vorhanden
 - Links zeigen zur Seite mit allen Usages
 - Usages haben `gridware_usage_id`

Erfolgskriterium: Ladevorgänge sind synchronisiert.

Stammdatenverwaltung

TC 4.1: Gridware Party Synchronisation

Ziel: Stelle sicher, dass Parteien aus Gridware korrekt nach ERPNext synchronisiert werden.

Testschritte:

1. Öffne Contract aus TC 5.1
2. Wähle entweder einen Customer oder den Supplier
3. Öffne die Gridware Party
4. **Validieren:**
 - Gridware Party wurde in ERPNext erstellt (Name: "GDW-P-.....")
 - display_name ist korrekt (z.B. "Tesla GmbH")
 - is_customer = 1 oder is_supplier = 1 gesetzt
 - gridware_customer_number / gridware_supplier_number sind gefüllt

Erfolgskriterium: Party-Datensatz ist vollständig mit allen Daten aus Gridware.

TC 4.2: Customer Auto-Erstellung aus Party

Ziel: Customer wird automatisch aus Gridware Party erstellt.

Testschritte:

1. Öffne die Gridware Party aus TC 5.1
2. Markiere: `is_customer = 1`
3. Speichere
4. Warte auf Background Job (oder triggere manuell: `master_data_from_gridware_party()`)
5. **Validieren:**
 - Ein neuer Customer wurde erstellt
 - Customer Name = Gridware Party display_name
 - `custom_gridware_party` Link ist gesetzt auf die Party
 - Kunde kann in Sales Invoice verwendet werden

Erfolgskriterium: Customer ist einsatzbereit ohne manuelle Nacharbeit.

TC 4.3: Contact Auto-Erstellung

Ziel: Kontaktperson wird automatisch aus Party erstellt.

Testschritte:

1. Erstelle neue Gridware Party mit `party_kind = "Person"`
2. Fülle: `first_name = "Max"`, `last_name = "Müller"`, `email = "max@example.com"`
3. Speichere
4. **Validieren:**
 - Ein Contact wurde erstellt
 - Contact Name = "Max Müller"
 - Email ist gespeichert
 - `custom_gridware_party` ist gesetzt

Erfolgskriterium: Contact kann für Customer verwendet werden.

TC 4.4: SEPA Mandate Synchronisation

Ziel: SEPA Mandat wird korrekt synchronisiert.

Testschritte:

1. Synchronisiere eine Party mit Payment Methods
2. Gehe zur Gridware Party → Tab "Payment Method"
3. **Validieren:**
 - Payment Method Row zeigt: `payment_type`, `iban`, `bic`, `sepa_mandate_id`
 - `erpnext_sepa_mandate` Link ist gesetzt
 - ERPNext SEPA Mandate wurde erstellt
 - Bank Account ist verlinkt

Erfolgskriterium: SEPA-Daten können in Sales Invoice verwendet werden.

Ladevorgang-Verarbeitung

TC 7.1: Usage Skeleton Creation

Ziel: Gridware Usage wird mit Status "Pending" erstellt.

Testschritte:

1. Öffne Gridware Usage List
2. Filtere nach: `status = "Pending"`

3. Öffne einen Usage

4. **Validieren:**

- gridware_usage_id ist eindeutig gefüllt
- gridware_contract ist verlinkt
- status = "Pending"
- Detail-Felder sind LEER: start, end, consumption_value_in_kwh, etc.

Erfolgskriterium: Skeleton-Format ist korrekt (nur ID + Contract).

TC 7.2: Usage Detail Fill (Async Job)

Ziel: Background Job füllt Usage-Details auf.

Testschritte:

1. Warte 1-2 Minuten nach TC 7.1

2. Aktualisiere den Usage (F5)

3. **Validieren:**

- status hat sich zu "Complete" geändert
- start ist gefüllt mit echtem Datetime
- end ist gefüllt
- consumption_value_in_kwh > 0
- usage_price_payment_amount > 0
- rental_object_display_name zeigt Ladesäule

Erfolgskriterium: Alle Daten aus Gridware API sind übernommen.

TC 7.3: Usage Price Details

Ziel: Kostenaufschlüsselung ist korrekt.

Testschritte:

1. Öffne einen "Complete" Usage

2. Gehe zu "Usage Price Details" Section

3. **Validieren:**

- $\text{total_fix_costs_excl_vat} + \text{total_energy_costs_excl_vat} + \dots = \text{Gesamtnetto}$
- Brutto-Werte sind ca. 19% höher (bei 19% VAT)

- Beispiel: 100 € netto + 19 € VAT = 119 € brutto ✓

Erfolgskriterium: Kostenaddition ist mathematisch korrekt.

TC 7.4: Usage Wh → kWh Conversion

Ziel: Energiewert wird korrekt von Wh zu kWh konvertiert.

Testschritte:

1. Öffne einen "Complete" Usage
2. Vergleiche: consumption_value_in_wh vs. consumption_value_in_kwh
3. **Validieren:**
 - $\text{consumption_value_in_kwh} = \text{consumption_value_in_wh} / 1000$
 - Beispiel: 45000 Wh = 45 kWh ✓

Erfolgskriterium: Umrechnung ist korrekt.

TC 7.5: Usage Cent → Euro Conversion

Ziel: Preise werden korrekt von Cent zu Euro konvertiert.

Testschritte:

1. Öffne einen "Complete" Usage
2. Vergleiche: usage_price_payment_amount Wert
3. **Validieren:**
 - Wenn API: 12500 Cent → DB: 125.00 € ✓
 - Wert ist in korrektem Format (2 Dezimalstellen)

Erfolgskriterium: Währungsumrechnung ist korrekt.

TC 7.6: Failed Usage Retry

Ziel: Fehlgeschlagene Usages werden täglich wiederholt.

Testschritte:

1. Erstelle einen simulierten Failed Usage (manuell: status = "Failed")
2. Warte auf Scheduler (täglich)
3. Oder: Triggere manuell `retry_failed_usage_completion()`
4. **Validieren:**
 - Status hat sich zu "Complete" geändert
 - Error Log zeigt keine Fehler mehr

Erfolgskriterium: Automatische Wiederholung funktioniert.

Kundenabrechnung

TC 8.1: Auto-Fetch beim Öffnen

Ziel: Customer Usage Data holt automatisch Usages ab.

Testschritte:

1. Erstelle neue Customer Usage Data
2. Setze: customer = "Customer A", billing_period = "01/2026"
3. Speichere
4. **Validieren:**
 - Ein Benachrichtigungs-Popup erscheint: "Auto-fetched X usage(s)"
 - usage_details Tabelle ist NICHT leer
 - Mindestens eine Zeile pro verfügbarem Usage

Erfolgskriterium: Auto-Fetch wurde triggert und hat Daten gefunden.

TC 8.2: Aggregation in usage_details

Ziel: Ladevorgänge werden korrekt in Child Table aggregiert.

Testschritte:

1. Öffne die Customer Usage Data aus TC 8.1

2. Schaue zu "Usage Details"

3. **Validieren:**

- Jede Zeile zeigt einen Usage
- gridware_usage Link ist gesetzt
- consumption_kwh ist > 0
- price_excl_vat ist > 0
- contract_type ist nicht leer

Erfolgskriterium: Alle Usages sind aggregiert.

TC 8.3: Item-Mapping

Ziel: Item Code wird korrekt aus Contract Type + VAT gemappt.

Testschritte:

1. Öffne Customer Usage Data aus TC 8.1
2. Schaue Spalte "item_code" in usage_details
3. **Validieren:**
 - Jede Zeile hat einen item_code (z.B. "CHARGING-POSTPAID-19")
 - item_code existiert in ERPNext Item Master
 - Mapping ist logisch: USAGE_POSTPAID + 19% = CHARGING-POSTPAID-19

Erfolgskriterium: Item-Mapping wurde korrekt durchgeführt.

TC 8.4: Totals Calculation

Ziel: Gesamtzahlen werden korrekt berechnet.

Testschritte:

1. Öffne Customer Usage Data aus TC 8.1
2. Gehe zu "Totals" Section
3. **Validieren (Beispiel mit 2 Usages):**
 - total_consumption = 45 + 30 = 75 kWh
 - total_charging_sessions = 2
 - total_price_excl_vat = SUM(price_excl_vat) von allen Details
 - total_price_incl_vat = total_price_excl_vat * 1.19 (bei 19% VAT)

Erfolgskriterium: Alle Summen sind mathematisch korrekt.

TC 8.5: Price Details Breakdown

Ziel: Kostenaufschlüsselung wird korrekt summiert.

Testschritte:

1. Öffne Customer Usage Data aus TC 8.1
2. Gehe zu "Price Details" Section
3. **Validieren:**
 - $\text{total_fix_costs_excl_vat} = \text{SUM aller Usage Fix Costs}$
 - $\text{total_energy_costs_excl_vat} = \text{SUM aller Usage Energy Costs}$
 - $\text{Summe aller Kosten} = \text{total_price_excl_vat}$

Erfolgskriterium: Kostenaufschlüsselung ist vollständig und korrekt.

TC 8.6: Duplicate Prevention

Ziel: Ein Usage kann nicht zweimal aggregiert werden.

Testschritte:

1. Erstelle eine zweite Customer Usage Data für denselben Customer/Period
2. **Validieren:**
 - Auto-Fetch findet KEINE Usages (weil bereits aggregiert)
 - Benachrichtigung: "No usages found"

Erfolgskriterium: Duplikate sind ausgeschlossen.

TC 8.7: Status Transition

Ziel: Status wird automatisch abgeleitet.

Testschritte:

1. Öffne Customer Usage Data aus TC 8.1
2. Status sollte "Uninvoiced" sein
3. Klicke "Create Sales Invoice"
4. Speichere Invoice als Draft
5. **Validieren:**
 - Status ändert sich zu "Draft Invoice"
6. Submit Invoice
7. **Validieren:**
 - Status ändert sich zu "Invoiced"

Erfolgskriterium: Status wird automatisch synchronisiert.

TC 8.8: Multiple Contracts Aggregation

Ziel: Mehrere Verträge werden korrekt aggregiert.

Voraussetzung: Customer hat 2 Contracts für Januar 2026

- Contract 1: USAGE_POSTPAID, 19% VAT
- Contract 2: AUTHORIZATION, 0% VAT

Testschritte:

1. Erstelle Customer Usage Data für diesen Customer
2. **Validieren:**
 - usage_details hat mindestens 2 unterschiedliche contract_type Zeilen
 - Zeilen sind nach Contract Type gruppiert

Erfolgskriterium: Mehrere Verträge sind gemeinsam abgerechnet.

TC 8.9: Sales Invoice Creation

Ziel: Sales Invoice wird aus Customer Usage Data erstellt.

Testschritte:

1. Öffne "Uninvoiced" Customer Usage Data
2. Klicke Button "Create Sales Invoice"

3. Validieren:

- Sales Invoice wird geöffnet (Draft Status)
- customer ist korrekt gesetzt
- Items sind gruppiert nach Contract Type + VAT
- Summe = total_price_excl_vat aus CUD
- custom_customer_usage_data Link ist gesetzt

Erfolgskriterium: Invoice ist korrekt vorbereitet und bereit zum Submit.

TC 8.10: Pending Usage Blockade

Ziel: Customer Usage Data kann nicht erstellt werden mit Pending Usages.

Testschritte:

1. Erstelle einen Usage mit status = "Pending"
2. Versuche Customer Usage Data zu erstellen
3. **Validieren:**
 - Error-Nachricht: "Found X pending Gridware Usage record(s)"
 - Customer Usage Data wird NICHT erstellt

Erfolgskriterium: Pending Usages werden blockiert.

Integrations-Tests (End-to-End)

E2E TC 1: Vollständiger Flow (Single Charging Session)

Szenario: Ein Ladevorgang eines Kunden in Januar 2026

Testschritte:

1. **Party Sync:** Synchronisiere Party (Kunde + Lieferant)
 - Party vorhanden

2. **Contract Sync:** Synchronisiere Vertrag
 - Contract vorhanden mit Kunden + Lieferant
3. **Usage Sync:** Warte auf Usage (Status: Pending → Complete)
 - Usage vorhanden mit allen Details
4. **Auto-Fetch:** Erstelle Customer Usage Data
 - Usage wurde aggregiert
5. **Totals:** Prüfe Gesamtzahlen
 - total_consumption = 45 kWh
 - total_price_excl_vat = 105.04 €
6. **Invoice:** Erstelle Sales Invoice
 - Invoice zeigt 45 kWh
 - Subtotal = 105.04 €
 - Total (mit 19% VAT) = 125.00 €

Erfolgskriterium: Vollständiger Flow ohne Fehler.

E2E TC 2: Mehrere Ladevorgänge (Monthly Aggregation)

Szenario: 100 Ladevorgänge eines Kunden in Januar 2026

Testschritte:

1. Synchronisiere alle 100 Usages
2. Erstelle Customer Usage Data für Januar
3. **Validieren:**
 - Alle 100 Usages sind aggregiert
 - total_charging_sessions = 100
 - total_consumption = 1.234 kWh (Beispiel)
 - Keine Duplikate
4. Erstelle Sales Invoice
5. **Validieren:**
 - Invoice zeigt alle Kosten
 - Gruppiert nach Contract Type + VAT (z.B. 2-3 Zeilen)

Erfolgskriterium: Massenaggregation funktioniert.

E2E TC 3: Multiple Contracts Same Customer

Szenario: Kunde hat 3 Verträge, Ladevorgänge pro Vertrag

Testschritte:

1. Synchronisiere 3 Contracts für denselben Kunden
2. Synchronisiere Usages pro Contract
3. Erstelle Customer Usage Data
4. **Validieren:**
 - Alle Contracts sind aggregiert
 - Customer Usage Detail hat Zeilen pro Contract Type
 - Totals addieren sich korrekt
5. Erstelle Sales Invoice
6. **Validieren:**
 - Invoice hat separate Zeilen pro Contract Type
 - Gesamtbetrag ist korrekt

Erfolgskriterium: Multi-Contract-Aggregation funktioniert.

E2E TC 4: USAGE_ADHOC Handling

Szenario: Kunde hat auch USAGE_ADHOC Ladevorgänge

Testschritte:

1. Synchronisiere Contract mit type = "USAGE_ADHOC"
2. Synchronisiere Usages
3. Erstelle Customer Usage Data
4. **Validieren:**
 - USAGE_ADHOC Usages sind NICHT aggregiert
 - Total enthält NICHT die USAGE_ADHOC Kosten
5. Erstelle Sales Invoice
6. **Validieren:**
 - USAGE_ADHOC Zeilen sind NICHT in Invoice

Erfolgskriterium: USAGE_ADHOC wird korrekt ignoriert.

E2E TC 5: Fehlerbehandlung & Recovery

Szenario: API ist kurzzeitig nicht erreichbar

Testschritte:

1. Trenne API-Verbindung
2. Starte Usage Sync
3. **Validieren:**
 - Usage Sync schlägt fehl
 - Status = "Failed"
 - Error Log zeigt API-Fehler
4. Stelle API-Verbindung wieder her
5. Warte auf Scheduler oder trigger `retry_failed_usage_completion()`
6. **Validieren:**
 - Status ändert sich zu "Complete"
 - Usage ist jetzt verfügbar

Erfolgskriterium: Error Recovery funktioniert.

E2E TC 6: SEPA Payment Integration

Szenario: Rechnung wird via SEPA bezahlt

Testschritte:

1. Erstelle und submit Sales Invoice (aus TC 8.9)
2. **Validieren:**
 - custom_sepa_mandate_reference ist gesetzt vom Customer
3. Erstelle Payment Entry
4. **Validieren:**
 - SEPA-Konto ist vorgewählt
 - Mandat ist korrekt verlinkt
5. Submit Payment Entry
6. **Validieren:**
 - Invoice ist bezahlt
 - Status-Update

Erfolgskriterium: SEPA-Integration funktioniert end-to-end.

Performance-Tests

PT 1: Bulk Sync (1.000+ Usages)

Ziel: System verarbeitet große Mengen.

Testschritte:

1. Synchronisiere Contract mit 1.000+ Usages
2. **Validieren:**
 - Sync fertig in <5 Minuten
 - Keine Datenverluste
 - Keine Duplikate

Erfolgskriterium: Performance ist akzeptabel.

PT 2: Aggregation Performance (100+ Sessions pro Customer)

Ziel: Customer Usage Data verarbeitet große Datenmengen.

Testschritte:

1. Erstelle Customer Usage Data mit 100+ Usages
2. **Validieren:**
 - Aggregation fertig in <10 Sekunden
 - Totals korrekt
 - UI responsive

Erfolgskriterium: UI bleibt responsive.

Data Quality Tests

DQ 1: Datenintegrität

Testschritte:

1. Exportiere Customer Usage Data
 2. Vergleiche mit Gridware Export
 3. **Validieren:**
 - Alle Usages vorhanden
 - Keine fehlenden oder doppelten Daten
 - Preise stimmen überein
-

DQ 2: Zahlen-Validierung

Testschritte:

1. Öffne mehrere Sales Invoices
 2. **Validieren:**
 - Jede Invoice = genau 1 Customer Usage Data
 - Summen stimmen überein
 - Keine mathematischen Fehler
-

Checkliste für Kundenabnahme

- Alle TC 4.x (Stammdaten) bestanden
- Alle TC 5.x (Vertragsmanagement) bestanden
- Alle TC 7.x (Ladevorgang-Verarbeitung) bestanden
- Alle TC 8.x (Kundenabrechnung) bestanden
- Alle E2E TCs bestanden
- Alle PT TCs bestanden
- Alle DQ TCs bestanden

- Keine kritischen Error-Logs
- Performance-Anforderungen erfüllt
- Go-Live freigegeben



Revision #1

Created 2026-07-24 14:31:57 CEST by Celia Lohner

Updated 2026-07-24 14:57:29 CEST by Celia Lohner