Table of Contents

Directo API Documentation

Welcome to the Directo API - a comprehensive RESTful API for accessing and managing your ERP data including items, orders, customers, invoices, stock levels, and more.

Getting Started

Prerequisites

- Active Directo ERP account
- API key (obtain from your Directo administrator)
- Basic understanding of REST APIs and HTTP methods
- API Resources and Endpint can be found in Postman collection. Response field list managed in Directo ERP. Postman collection can be downloaded here: POSTMAN

Quick Start Guide

1. **Configure Authentication**: Set your API key in the collection's Authorization tab or update the X-Directo-Key header
2. **Set Variables**: Configure baseUrl and version in collection variables
3. **Choose Format**: Set the Accept header to application/json or application/xml
4. **Make Your First Request**: Try the „All items“ request to retrieve your item catalog

Base URL & Versioning

Base URL Structure:

```
{{baseUrl}}/v{{version}}/{resource}
```

Current Configuration:

- **baseUrl**: https://login.directo.ee/apidirect
- **version**: 1 (current API version)

Example Full URL:

```
[https://login.directo.ee/apidirect/v1/items](https://login.directo.ee/apidirect/v1/items)
```

The API uses versioning to ensure backward compatibility. Always specify the version number in your requests.

Authentication

The Directo API uses API Key authentication for secure access.

How to Authenticate

Include your API key in the request header:

```
X-Directo-Key: YOUR_API_KEY_HERE
```

Getting Your API Key

Contact your Directo system administrator to obtain an API key. Each key is associated with specific permissions and access levels.

Security Best Practices

- Never expose your API key in client-side code or public repositories
- Store API keys securely using environment variables
- Rotate keys periodically
- Use different keys for development and production environments

Response Formats (JSON vs XML)

The API supports both JSON and XML response formats. Control the format using the Accept header.

JSON Format (Recommended)

```
Accept: application/json
```

Example Request:

```
curl -X GET "{{baseUrl}}/v{{version}}/items"

-H "X-Directo-Key: YOUR_API_KEY"

-H "Accept: application/json"
```

XML Format

```
Accept: application/xml
```

Example Request:

```
curl -X GET "{{baseUrl}}/v{{version}}/items"

-H "X-Directo-Key: YOUR_API_KEY"

-H "Accept: application/xml"
```

Advanced Filtering (Key Feature)

ERP-Style Filtering - The Power of Directo API

CRITICAL FEATURE: The Directo API supports the exact same powerful filtering capabilities as the Directo ERP system itself.

This is one of the most powerful features of the Directo API. Filters can be applied like default filters inside the ERP system. All fields can be used for filters except calculated fields.

What This Means for Developers

- **Seamless Integration:** Any filter that works in your Directo ERP interface can be applied via the API using query parameters
- **No Learning Curve:** If you know how to filter in the ERP, you already know how to filter via the API
- **Maximum Flexibility:** Filter by ANY field in the response data (except calculated fields)
- **Powerful Queries:** Combine multiple filters to create complex queries without custom endpoints

Understanding Filterable vs Calculated Fields

Filterable Fields (Stored in Database) These are fields that are stored directly in the database and can be used for filtering:

- **Item Fields:** code, name, class, status, country, unit, price, cost, supplier_code, barcode, weight, volume
- **Order/Invoice Fields:** number, date, status, customer_code, customer_name, total, currency, payment_terms, delivery_date, warehouse
- **Customer Fields:** code, name, country, city, address, phone, email, vat_number, payment_terms, credit_limit
- **Stock Fields:** item_code, warehouse, quantity, reserved_quantity, available_quantity, location, batch_number, serial_number
- **Line Item Fields:** row_item, row_quantity, row_price, row_description, row_discount, row_warehouse
- **Custom Data Fields:** Any custom fields you've defined in your ERP system (found in datafields array)

Calculated Fields (Cannot Be Filtered) These are fields computed on-the-fly and cannot be used

for filtering:

- **Computed Totals:** Fields like total_with_vat, line_total, discount_amount (when calculated from other fields)
- **Derived Values:** available_stock (when calculated as quantity - reserved), profit_margin (calculated from price and cost)
- **Aggregations:** total_orders, average_price, sum_quantity (when aggregated from multiple records)
- **Formatted Values:** Display-only fields like formatted_date, formatted_price (when they're just formatting of other fields)
- **Virtual Fields:** Fields that exist only in the API response but not in the database

How to Identify Calculated Fields

- **Check the ERP Interface:** If you cannot filter by a field in the ERP system's filter dialog, it's likely calculated
- **Test the Filter:** Try filtering by the field via API - if it returns no results or an error, it's calculated
- **Look for Patterns:** Fields with names like total_*, calculated_*, formatted_*, or display_* are often calculated
- **Consult Documentation:** Check your ERP system's field documentation or ask your administrator

Practical Examples: Filterable vs Calculated

Example 1: Item Stock

```
 WORKS: GET /items?quantity=>100
(quantity is stored in database)
 DOESN'T WORK: GET /items?available_stock=>100
(if available_stock = quantity - reserved, it's calculated)
```

Example 2: Order Totals

```
 WORKS: GET /orders?row_price=>1000
(row_price is stored per line item)
 MIGHT NOT WORK: GET /orders?order_total=>5000
(if order_total is sum of all line items, it's calculated)
 ALTERNATIVE: GET /orders?date=>2025-01-01&status=COMPLETED
(then calculate totals in your application)
```

Example 3: Customer Data

```
 WORKS: GET /customers?country=EE&city=Tallinn
(both are stored fields)
 DOESN'T WORK: GET /customers?full_address=*Tallinn*
(if full_address is concatenation of address+city+country)
 ALTERNATIVE: GET /customers?city=Tallinn
```

```
(filter by individual stored fields)
```

Basic Filtering

Filter by exact match using any stored field name:

```
GET {{baseUrl}}/v{{version}}/items?code=9810
GET {{baseUrl}}/v{{version}}/items?status=UUS
GET {{baseUrl}}/v{{version}}/orders?number=12345
GET {{baseUrl}}/v{{version}}/customers?country=EE
GET {{baseUrl}}/v{{version}}/items?class=KAUP
```

Key Principle: Use the exact field name as it appears in the API response (or as you see it in the ERP system).

Contains Filtering

This filtering allows you to search for values containing a specific substring.

This works like SQL LIKE `'%text%'` logic. Use it when you need flexible filtering for names, codes, descriptions, or any other text fields

```
GET {{baseUrl}}/v{{version}}/customers?name=%john%
```

Comparison Operators

The API supports powerful comparison operators for advanced filtering:

Greater Than (>)

```
GET {{baseUrl}}/v{{version}}/orders?date=>2025-09-01T00:00:00
```

```
GET {{baseUrl}}/v{{version}}/invoices?row_quantity=>5000
```

Less Than (<)

```
GET {{baseUrl}}/v{{version}}/orders?date=<2025-09-03T00:00:00
```

```
GET {{baseUrl}}/v{{version}}/invoices?row_quantity=<5000
```

Not Equal (!)

```
GET {{baseUrl}}/v{{version}}/items?country=!EE
```

```
GET {{baseUrl}}/v{{version}}/orders?status=!CLOSED
```

Greater Than or Equal (>=)

```
GET {{baseUrl}}/v{{version}}/items?price=>100
```

```
GET {{baseUrl}}/v{{version}}/orders?date=>2025-01-01T00:00:00
```

Less Than or Equal (< =)

```
GET {{baseUrl}}/v{{version}}/items?weight=<5.5
```

```
GET {{baseUrl}}/v{{version}}/orders?date=<2025-12-31T23:59:59
```

Range Filtering

Combine operators to create ranges (AND logic):

```
GET
{{baseUrl}}/v{{version}}/orders?date=>2025-09-01T00:00:00&date=<2025-09-03T0
0:00:00
GET {{baseUrl}}/v{{version}}/items?quantity=>10&quantity=<100
```

Multiple Values (OR Logic)

Use comma-separated lists to filter by multiple values:

```
GET {{baseUrl}}/v{{version}}/items?warehouse=AALTR,AIPM,WAREHOUSE1
GET {{baseUrl}}/v{{version}}/orders?status=NEW,PENDING,PROCESSING
GET {{baseUrl}}/v{{version}}/customers?country=EE,LV,LT
```

Timestamp-Based Filtering (Synchronization)

Use the ts parameter to retrieve records modified after a specific timestamp (ideal for synchronization):

```
GET {{baseUrl}}/v{{version}}/items?ts=>2025-10-01T08:11:05.129Z
GET {{baseUrl}}/v{{version}}/customers?ts=>2025-01-01T00:00:00Z
```

Best Practice: Store the timestamp of your last successful sync and use it in the next request to fetch only changed records.

Filtering by Line Item Fields

For resources with line items (orders, invoices), you can filter by line-level fields:

```
GET {{baseUrl}}/v{{version}}/orders?row_item=343443
```

```
GET {{baseUrl}}/v{{version}}/invoices?row_quantity=>10
```

Note: When filtering by line item fields, the API returns the entire document (order/invoice) if ANY line matches the filter.

Filtering by Custom Data Fields

If you have custom fields defined in your ERP system (visible in the datafields array), you can filter by them:

```
GET {{baseUrl}}/v{{version}}/items?IS_WEIGHT=1
GET {{baseUrl}}/v{{version}}/items?VISUAL_VERIFICATION=1
```

How to Find Custom Field Names: Make a request without filters and examine the datafields array in the response.

Complex Filter Combinations

Combine multiple filters to create sophisticated queries:

Get new orders from the last week for a specific customer:

```
GET
{{baseUrl}}/v{{version}}/orders?status=NEW&date=>2025-11-13T00:00:00&custome
r_code=CUST001
```

Get items with low stock in specific warehouses, excluding closed items:

```
GET
{{baseUrl}}/v{{version}}/items?quantity=<10&warehouse=AALTR,AIPM&status=!CLO
SED
```

Common Filter Parameters by Resource

Parameter	Description	Example	Applicable Resource
ts	Timestamp for modified records (ISO 8601)	2025-10-01T08:11:05.129Z	All resources
date	Date filtering (supports operators)	>2025-09-01T00:00:00	Orders, Invoices
number	Document number	12345 or !001	Orders, Invoices
code	Item/Customer code	9810 or CUST001	Items, Customers
name	Name search	Kalev	Items, Customers
status	Status filter	UUS, CLOSED, NEW	Orders, Items
country	Country code (ISO 2-letter)	EE, !EE	Items, Customers
warehouse	Warehouse/stock location code	AALTR,AIPM	Items, Stock Levels, Orders
quantity	Quantity filter (supports operators)	>100, <50	Items, Stock Levels
row_item	Item code in line items	9810 or PROD-ABC	Orders, Invoices

Parameter	Description	Example	Applicable Resource
closed	Closed status	1 (closed), 0 (open)	Customers, Orders

Filter Best Practices

- **Start Simple:** Begin with single filters and add complexity as needed
- **Use Timestamp Sync:** Always use ts parameter for incremental synchronization
- **Combine Strategically:** Use multiple filters to reduce data transfer and processing
- **Test in ERP First:** If unsure whether a field is filterable, test it in the ERP interface first
- **Handle Empty Results:** Empty results might mean filters are too restrictive or field is calculated
- **URL Encode:** Always URL-encode filter values, especially dates and special characters

Troubleshooting Filters

Problem: Filter returns no results

- Check: Is the field calculated? Try filtering by underlying stored fields
- Check: Is the value correct? Verify exact field values in an unfiltered response
- Check: Are operators correct? Use > not >= if that's what the ERP uses

Problem: Filter seems to be ignored

- Check: Is the field name spelled correctly? Field names are case-sensitive
- Check: Is the value URL-encoded? Special characters must be encoded
- Check: Does the field exist in this resource? Not all fields are available in all resources

Problem: Unexpected results

- Check: Are you using OR logic (comma) when you meant AND (multiple parameters)?
- Check: For line item filters, remember the entire document is returned if any line matches

API Resources & Endpoints

API Resources and Endpint can be found in Postman collection. Response field list managed in Directo ERP.

```
GET {{baseUrl}}/v{{version}}/sales_quotations
GET {{baseUrl}}/v{{version}}/orders
GET {{baseUrl}}/v{{version}}/invoices
GET {{baseUrl}}/v{{version}}/purchase_orders
GET {{baseUrl}}/v{{version}}/stock_receipts
GET {{baseUrl}}/v{{version}}/purchase_invoices
GET {{baseUrl}}/v{{version}}/purchase_payments
GET {{baseUrl}}/v{{version}}/customers
GET {{baseUrl}}/v{{version}}/suppliers
GET {{baseUrl}}/v{{version}}/contacts
GET {{baseUrl}}/v{{version}}/items
```

```
GET {{baseUrl}}/v{{version}}/itemclasses
GET {{baseUrl}}/v{{version}}/priceformulas
GET {{baseUrl}}/v{{version}}/stocklevels
GET {{baseUrl}}/v{{version}}/stocklevels_sn
GET {{baseUrl}}/v{{version}}/allocations
GET {{baseUrl}}/v{{version}}/accounts
GET {{baseUrl}}/v{{version}}/transactions
GET {{baseUrl}}/v{{version}}/deleted
GET {{baseUrl}}/v{{version}}/financial_budgets
GET {{baseUrl}}/v{{version}}/events
GET {{baseUrl}}/v{{version}}/receipts
GET {{baseUrl}}/v{{version}}/deliveries
GET {{baseUrl}}/v{{version}}/movements
GET {{baseUrl}}/v{{version}}/objects
GET {{baseUrl}}/v{{version}}/projects
GET {{baseUrl}}/v{{version}}/resources
GET {{baseUrl}}/v{{version}}/sales_contracts
GET {{baseUrl}}/v{{version}}/replacementlogs
```

Postman collection can be downloaded here: e: Directo API postman collection

Response Structure

Success Response (HTTP 200)

JSON Format:

```
[
{
"code": "1011",
"class": "KAUP",
"name": "Kommid Lily, Kalev",
"price": 12.50,
"quantity": 150,
"warehouse": "AALTR",
"datafields": [
{
"code": "TEST_ARTIKKEL",
"content": "Test lisaväli sisu"
},
{
"code": "IS_WEIGHT",
"content": "0"
},
{
"code": "VISUAL_VERIFICATION",
"content": "0"
```

```
},
{
"code": "VENSAFE",
"content": "0"
}
]
},
{
"code": "1012",
"class": "KAUP",
"name": "Šokolaad Kalev",
"price": 8.75,
"quantity": 200,
"warehouse": "AIPM",
"datafields": []
}
]
```

XML Format:

```
<items>
<item code="1011" class="KAUP" name="Kommid Lily, Kalev" price="12.50"
quantity="150" warehouse="AALTR">
<datafields>
<data code="TEST_ARTIKKEL" content="Test lisaväli sisu"/>
<data code="IS_WEIGHT" content="0"/>
<data code="VISUAL_VERIFICATION" content="0"/>
<data code="VENSAFE" content="0"/>
</datafields>
</item>
<item code="1012" class="KAUP" name="Šokolaad Kalev" price="8.75"
quantity="200" warehouse="AIPM">
<datafields/>
</item>
</items>
```

Response Characteristics

- **Array Format:** All endpoints return arrays of objects, even for single results
- **Consistent Structure:** Field names are consistent across all response formats
- **Custom Fields:** Custom data fields appear in the datafields array
- **Null Values:** Missing or null values may be omitted from responses
- **Date Format:** All dates use ISO 8601 format (e.g., 2025-11-20T07:30:00Z)
- **Numeric Values:** Numbers are returned as numbers in JSON, as strings in XML

⚠ Error Handling

HTTP Status Codes

Status Code	Meaning	Description
200	OK	Request successful
400	Bad Request	Invalid parameters or malformed request
400	API not enababled	Resource not enabled for this API key
401	Unauthorized	Missing or invalid API key
403	Forbidden	API key lacks required permissions
404	Not Found	Resource or endpoint not found
429	Too Many Requests	Rate limit exceeded
500	Internal Server Error	Server-side error
503	Service Unavailable	API temporarily unavailable

Error Response Format

```
{
"error": {
"code": "INVALID_API_KEY",
"message": "The provided API key is invalid or expired",
"details": "Please check your API key and try again"
}
}
```

Common Error Scenarios

401 Unauthorized:

- Missing X-Directo-Key header
- Invalid or expired API key
- **Solution:** Verify your API key and ensure it's included in the header

403 Forbidden:

- API key lacks permissions for the requested resource
- **Solution:** Contact your administrator to grant necessary permissions

400 Bad Request:

- Invalid filter syntax
- Malformed date format (use ISO 8601: 2025-11-20T07:30:00Z)
- Filtering by calculated field
- **Solution:** Check filter parameters and date formats

404 Not Found:

- Incorrect endpoint URL
- Invalid API version
- **Solution:** Verify the endpoint path and version number

429 Too Many Requests:

- Rate limit exceeded
- **Solution:** Implement exponential backoff and reduce request frequency

500 Internal Server Error:

- Server-side processing error
- **Solution:** Retry after a delay; contact support if persistent

Rate Limiting & Performance

Rate Limits

While specific rate limits depend on your account tier, follow these guidelines:

- **Recommended:** Max 60 requests per minute
- **Burst Limit:** Short bursts of higher traffic may be allowed
- **Daily Limits:** May apply based on your subscription

Performance Best Practices

1. Use Timestamp Filtering for Sync Instead of fetching all records repeatedly, use the ts parameter:

```
GET {{baseUrl}}/v{{version}}/items?ts>=2025-11-20T07:00:00Z
```

* **Benefits:** Reduces data transfer, Faster response times, Lower server load, Stays within rate limits

2. Implement Pagination with Date Ranges For large datasets, break requests into smaller chunks:

```
# Week 1

GET {{baseUrl}}/v{{version}}/orders?date=>2025-11-01&date=<2025-11-08

# Week 2

GET {{baseUrl}}/v{{version}}/orders?date=>2025-11-08&date=<2025-11-15
```

3. Cache Static Data Cache data that changes infrequently:

- Item classes (cache for 24 hours)
- Price formulas (cache for 1 hour)
- Customer master data (cache for 6 hours)

4. Use Specific Filters Apply filters to retrieve only the data you need:

```
# Instead of fetching all items and filtering locally

GET {{baseUrl}}/v{{version}}/items

# Fetch only what you need

GET {{baseUrl}}/v{{version}}/items?warehouse=AALTR&quantity=>0
```

5. Implement Exponential Backoff For transient errors (429, 503), retry with increasing delays:

- 1st retry: wait 1 second
- 2nd retry: wait 2 seconds
- 3rd retry: wait 4 seconds
- 4th retry: wait 8 seconds

6. Batch Related Requests Group related operations to minimize round trips:

```
1. Fetch orders: ''GET /orders?ts=>{last_sync}''
2. For each order, fetch customer details (if needed)
3. Update local database in batch
```

7. Monitor API Usage Track your API consumption:

- Log request counts per endpoint
- Monitor response times
- Track error rates
- Set up alerts for rate limit warnings

Common Integration Patterns

Pattern 1: Initial Data Sync (First-Time Setup)

Objective: Load all data from Directo ERP into your system

- **Step 1:** Fetch all items `GET /items`
- **Step 2:** Fetch all customers `GET /customers`
- **Step 3:** Fetch all stock levels `GET /stocklevels`
- **Step 4:** Fetch item classes `GET /itemclasses`
- **Step 5:** Store current timestamp for future incremental syncs `timestamp = current_time()`

Considerations: May take several minutes, run during off-peak hours, handle timeouts.

Pattern 2: Incremental Sync (Ongoing Synchronization)

Objective: Keep your system up-to-date with changes in Directo ERP

- **Step 1:** Retrieve last sync timestamp from your database

- **Step 2:** Fetch changed items `GET /items?ts⇒{last_sync}`
- **Step 3:** Fetch changed customers `GET /customers?ts⇒{last_sync}`
- **Step 4:** Fetch changed stock levels `GET /stocklevels?ts⇒{last_sync}`
- **Step 5:** Fetch deleted records `GET /deleted?ts⇒{last_sync}`
- **Step 6:** Process changes (Update/Insert/Delete)
- **Step 7:** Update last sync timestamp

Pattern 3: Real-Time Order Monitoring

Objective: Monitor new orders for immediate processing

- **Step 1:** Poll orders endpoint `GET /orders?status=NEW&ts⇒{last_check}`
- **Step 2:** Process new orders (Validate, Check inventory, Update status)
- **Step 3:** Update last check timestamp
- **Step 4:** Wait for next interval (e.g., 5 minutes)

Pattern 4: Inventory Management & Availability Check

Objective: Maintain accurate inventory availability

- **Step 1:** Fetch current stock levels `GET /stocklevels?item_code={sku}`
- **Step 2:** Fetch allocations (reserved stock) `GET /allocations?item_code={sku}`
- **Step 3:** Calculate available quantity: `available = stock_quantity - allocated_quantity`
- **Step 4:** Update your inventory system

Pattern 5: Customer Data Synchronization (CRM Integration)

Objective: Keep customer data synchronized between Directo and your CRM

- **Step 1:** Fetch changed customers `GET /customers?ts={last_sync}`
- **Step 2:** Sync to CRM (Create/Update)
- **Step 3:** Fetch deleted customers `GET /deleted?resource_type=customers&ts={last_sync}`
- **Step 4:** Remove deleted customers from CRM

Pattern 6: Invoice Export to Accounting System

Objective: Export invoices to external accounting software

- **Step 1:** Fetch new/updated invoices `GET /invoices?ts⇒{last_sync}`
- **Step 2:** Map fields and create in accounting system
- **Step 3:** Handle errors and mark as exported

Pattern 7: Price Synchronization to E-Commerce

Objective: Keep product prices updated in your online store

- **Step 1:** Fetch changed items with pricing `GET /items?ts⇒{last_sync}`
- **Step 2:** Fetch price formulas `GET /priceformulas?ts⇒{last_sync}`
- **Step 3:** Calculate final prices and update e-commerce platform

Troubleshooting Guide

Issue: Empty Response / No Results

Possible Causes: Filters too restrictive, incorrect date format, filtering by calculated field.
Solutions:

```
# Remove filters one by one

GET /items

# Verify date format

 CORRECT: ?date=>2025-11-20T00:00:00Z
```

Issue: Authentication Errors (401/403)

Possible Causes: Missing key, incorrect header name, expired key. **Solutions:**

```
# Verify header name (case-sensitive)

 CORRECT: X-Directo-Key: YOUR_KEY
```

Issue: Rate Limit Errors (429)

Solutions (Python):

```
import time
def make_request_with_retry(url, max_retries=5):
for attempt in range(max_retries):
response = requests.get(url)
if response.status_code == 200:
return response
elif response.status_code == 429:
wait_time = 2 ** attempt
time.sleep(wait_time)
raise Exception("Max retries exceeded")
```

Support & Resources

Technical Support:

- Email: info@directo.ee
- Contact your Directo account manager
- Submit support ticket through Directo Chat

Changelog & Version History

Version 1 (Current)

- Initial API release
- Support for major resources: items, customers, orders, invoices, stock levels
- JSON/XML support & ERP-style filtering

Security & Compliance

Data Security

- **Encryption**: All API requests must use HTTPS
- **Authentication**: API key required for all requests
- **Authorization**: Role-based access control via API keys
- **Audit Logging**: All API requests are logged server-side

Compliance Considerations

- GDPR: Customer data includes personal information - handle according to GDPR requirements
- Data Retention: Implement appropriate data retention policies
- Access Control: Limit API key distribution to authorized personnel only
- Monitoring: Monitor API usage for unusual patterns

Security Best Practices

- Protect API Keys:
  - Never commit keys to version control
  - Use environment variables or secure vaults
  - Rotate keys regularly (quarterly recommended)
  - Use different keys for dev/staging/production
- Implement Access Controls:
  - Limit API key permissions to minimum required
  - Use separate keys for different integrations
  - Revoke keys immediately when no longer needed
- Monitor Usage:
  - Log all API requests in your application

    - Set up alerts for unusual activity
    - Review access logs regularly
- Handle Data Responsibly:
    - Encrypt sensitive data at rest
    - Implement data retention policies
    - Comply with privacy regulations (GDPR, etc.)
    - Secure data transmission (HTTPS only)

Tips & Tricks

- **Tip 1:** Discover fields by making an unfiltered request: `GET /items?code=1011`
- **Tip 2:** Test filters in ERP interface first.
- **Tip 3:** Build filter strings programmatically.

Example (Python):

```
# Python example with comprehensive error handling
import requests
import time
def fetch_directo_data(endpoint, params=None, max_retries=3):
    url = f"{{{{baseUrl}}}}/v{{{{version}}}}/{endpoint}"
    headers = {
        "X-Directo-Key": "YOUR_API_KEY",
        "Accept": "application/json"
    }
    for attempt in range(max_retries):
        try:
            response = requests.get(url, headers=headers, params=params,
timeout=30)
            if response.status_code == 200:
                return response.json()
            elif response.status_code == 429:
                # Rate limit - wait and retry
                wait_time = 2 ** attempt
                time.sleep(wait_time)
            elif response.status_code in [401, 403]:
                # Auth error - don't retry
                raise Exception(f"Authentication error:
{response.status_code}")
            else:
                # Other error - log and retry
                print(f"Error {response.status_code}: {response.text}")
                time.sleep(1)
        except requests.exceptions.Timeout:
            print(f"Timeout on attempt {attempt + 1}")
            time.sleep(2)
        except requests.exceptions.RequestException as e:
            print(f"Request failed: {e}")
            time.sleep(2)
    raise Exception(f"Failed after {max_retries} attempts")
```

```
# Usage
items = fetch_directo_data("items", params={"warehouse": "AALTR",
"quantity": ">0"})
```

Last Updated: 29-01-2026 **API Version:** 1

Quick Reference Card

Essential Endpoints

```
Items:        GET /v1/items
Customers:    GET /v1/customers
Orders:       GET /v1/orders
Invoices:     GET /v1/invoices
Stock:        GET /v1/stocklevels
Deletions:    GET /v1/deleted
```

Authentication

```
X-Directo-Key: YOUR_API_KEY
```

Response Format

```
Accept: application/json  (or application/xml)
```

Common Filters

```
Exact match:      ?field=value
Greater than:     ?field=>value
Less than:        ?field=<value
Not equal:        ?field=!value
Multiple values:  ?field=value1,value2
Timestamp:        ?ts=2025-11-20T07:00:00Z
```

Filter Examples

```
?warehouse=AALTR
?quantity=>100
?date=>2025-11-01T00:00:00&date=<2025-12-01T00:00:00
?status=NEW,PENDING
?country=!EE
?ts=2025-11-20T07:00:00Z
```

Remember

- □ All fields can be filtered (except calculated fields)
- □ Filters mirror ERP system behavior
- □ Use timestamp filtering for sync
- □ URL encode special characters
- □ Implement rate limiting (60 req/min)
- □ Handle errors with exponential backoff

This documentation is maintained by the Directo development team. For questions or feedback, contact info@directo.ee