Sales Channel Integration API Documentation

Centre Research Peptides - External Order Creation API

Version: 1.0.0
Last Updated: January 17, 2026

Table of Contents

Overview

The Sales Channel Integration API allows external partners to programmatically create orders in the Centre Research system using API key authentication. This enables seamless integration with partner e-commerce platforms, marketplaces, and dropshipping systems.

Key Features

- **API Key Authentication** - Simple and secure authentication using API keys
- **Automatic Customer Management** - Customers are automatically created or matched by email
- **Channel-Based Pricing** - Prices are automatically applied from your sales channel configuration
- **Idempotency** - Prevent duplicate orders using unique partner order IDs
- **Real-time Order Creation** - Orders are created instantly and ready for fulfillment

Important Notes

⚠️ **Pricing**: The system uses ONLY the prices configured in your sales channel's price list. Any prices sent in the request payload are ignored.

⚠️ **Product Availability**: All product variants must have a price configured in your channel's price list, otherwise the request will fail.

⚠️ **Idempotency**: Use unique `partnerOrderId` values to prevent duplicate order creation.

Authentication

All API requests must include an API key in the request headers for authentication.

Headers Required

```
Content-Type: application/json
X-API-Key: your-api-key-here
```

Obtaining Your API Key

1. Contact Centre Research administration to set up your sales channel account
2. Your unique API key will be generated automatically when your sales channel is created
3. Store your API key securely - it provides full access to create orders on your behalf
4. Never share your API key or commit it to version control

Security Best Practices

- Store API keys in environment variables or secure key management systems
- Rotate API keys periodically
- Use HTTPS for all API requests
- Monitor API usage for suspicious activity

API Endpoint

Base URL

```
https://pepapi.stmin.dev/api
```

Create Order Endpoint

```
POST /sales-channels/integration/orders
```

Full URL:

```
https://pepapi.stmin.dev/api/sales-channels/integration/orders
```

Request Format

HTTP Method

`POST`

Headers

```
Content-Type: application/json
X-API-Key: your-api-key-here
```

Request Body Schema

```
{
  "partnerOrderId": "string (required)",
  "customer": {
    "firstName": "string (required)",
```

```
    "lastName": "string (required)",
    "email": "string (required, must be valid email)",
    "phone": "string (optional)"
  },
  "shippingAddress": {
    "firstName": "string (required)",
    "lastName": "string (required)",
    "address1": "string (required)",
    "address2": "string (optional)",
    "city": "string (required)",
    "state": "string (required, 2-letter state code)",
    "zip": "string (required, postal code)",
    "country": "string (required, default: US)"
  },
  "billingAddress": {
    "firstName": "string (optional)",
    "lastName": "string (optional)",
    "address1": "string (optional)",
    "address2": "string (optional)",
    "city": "string (optional)",
    "state": "string (optional, 2-letter state code)",
    "zip": "string (optional, postal code)",
    "country": "string (optional)"
  },
  "items": [
    {
      "variantId": "string (required, variant ID or SKU)",
      "quantity": "integer (required, minimum: 1)"
    }
  ]
}
```

Field Descriptions

partnerOrderId

- **Type**: String
- **Required**: Yes
- **Description**: Your unique order identifier from your system
- **Purpose**: Used for idempotency to prevent duplicate orders
- **Constraints**: Must be unique across all orders for your sales channel
- **Example**: `"PARTNER-20260117-001"`

customer

- **Type**: Object
- **Required**: Yes
- **Description**: Customer information for the order

Customer Fields:

Field	Type	Required	Description
firstName	String	Yes	Customer's first name

lastName	String	Yes	Customer's last name
email	String	Yes	Valid email address (used for customer matching)
phone	String	No	Contact phone number

Note: If a customer with the provided email already exists, their existing record will be used. Otherwise, a new customer record is created.

shippingAddress

- **Type**: Object
- **Required**: Yes
- **Description**: Destination address for shipment

Shipping Address Fields:

Field	Type	Required	Description
firstName	String	Yes	Recipient's first name
lastName	String	Yes	Recipient's last name
address1	String	Yes	Street address line 1
address2	String	No	Street address line 2 (apartment, suite, etc.)
city	String	Yes	City name
state	String	Yes	2-letter US state code (e.g., "CA", "NY")
zip	String	Yes	Postal/ZIP code
country	String	Yes	Country code (default: "US")

billingAddress

- **Type**: Object
- **Required**: No
- **Description**: Billing address for the order
- **Fallback**: If not provided, uses shippingAddress

Billing Address Fields: Same as shipping address fields

items

- **Type**: Array of Objects
- **Required**: Yes
- **Constraints**: Must contain at least 1 item
- **Description**: Products to be ordered

Item Fields:

Field	Type	Required	Description
variantId	String	Yes	Centre Research's variant ID **OR** unique SKU code
quantity	Integer	Yes	Quantity to order (minimum: 1)

Variant Identification (ID or SKU):

The `variantId` field accepts either:

1. **Variant ID** (CUID format): Centre Research's internal database ID (e.g., `"clxyz123abc456def789"`)
2. **SKU Code**: The unique product SKU code (e.g., `"BPC-157-5MG"`, `"TB500-10MG"`)

Lookup Priority:

1. The system first attempts to find the variant by database ID
2. If not found, it then searches by SKU code
3. If neither matches, an error is returned

Important: Each variant must have a configured price in your sales channel's price list, regardless of whether you use ID or SKU.

Response Format

Success Response

Status Code: `201 Created`

```
{
  "success": true,
  "data": {
    "orderId": "clp123abc456def789ghi",
    "orderNumber": "ORD-1705329600-ABC123",
    "status": "PENDING"
  }
}
```

Response Fields

Field	Type	Description
success	Boolean	Indicates if the request was successful
data.orderId	String	Centre Research's internal order ID (use for tracking)
data.orderNumber	String	Human-readable order number
data.status	String	Current order status (will be "PENDING" for new orders)

Example Requests

Example 1: Basic Single Item Order

```
curl -X POST https://pepapi.stmin.dev/api/sales-channels/integration/orders \
  -H "Content-Type: application/json" \
  -H "X-API-Key: your-api-key-here" \
  -d '{
```

```
    "partnerOrderId": "PARTNER-20260117-001",
    "customer": {
      "firstName": "John",
      "lastName": "Doe",
      "email": "john.doe@example.com",
      "phone": "555-0199"
    },
    "shippingAddress": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "123 Main Street",
      "city": "Los Angeles",
      "state": "CA",
      "zip": "90001",
      "country": "US"
    },
    "items": [
      {
        "variantId": "clxyz123abc456def789",
        "quantity": 2
      }
    ]
  }'
```

Expected Response:

```
{
  "success": true,
  "data": {
    "orderId": "clp123abc456def789ghi",
    "orderNumber": "ORD-1705329600-001",
    "status": "PENDING"
  }
}
```

Example 2: Multiple Items with Separate Billing Address

```
{
  "partnerOrderId": "PARTNER-20260117-002",
  "customer": {
    "firstName": "Jane",
    "lastName": "Smith",
    "email": "jane.smith@example.com",
    "phone": "555-0198"
  },
  "shippingAddress": {
    "firstName": "Jane",
    "lastName": "Smith",
    "address1": "456 Oak Avenue",
    "address2": "Apartment 3B",
```

```
    "city": "New York",
    "state": "NY",
    "zip": "10001",
    "country": "US"
  },
  "billingAddress": {
    "firstName": "Jane",
    "lastName": "Smith",
    "address1": "789 Business Park Drive",
    "city": "New York",
    "state": "NY",
    "zip": "10002",
    "country": "US"
  },
  "items": [
    {
      "variantId": "clxyz123abc456def789",
      "quantity": 1
    },
    {
      "variantId": "clxyz456def789ghi012",
      "quantity": 3
    },
    {
      "variantId": "clxyz789ghi012jkl345",
      "quantity": 5
    }
  ]
}
```

Example 3: Using SKU Codes Instead of Variant IDs

You can use product SKU codes instead of variant IDs for easier integration:

```
{
  "partnerOrderId": "PARTNER-20260117-003",
  "customer": {
    "firstName": "Robert",
    "lastName": "Johnson",
    "email": "robert.j@techcorp.com",
    "phone": "555-0197"
  },
  "shippingAddress": {
    "firstName": "Robert",
    "lastName": "Johnson",
    "address1": "1000 Technology Way",
    "address2": "Building B, Floor 5",
    "city": "San Francisco",
    "state": "CA",
    "zip": "94105",
    "country": "US"
```

```
  },
  "items": [
    {
      "variantId": "BPC-157-5MG",
      "quantity": 2
    },
    {
      "variantId": "TB500-10MG",
      "quantity": 1
    }
  ]
}
```

Note: The `variantId` field accepts either the database ID (e.g., `"clxyz789ghi012jkl345"`) or the product SKU (e.g., `"BPC-157-5MG"`). The system will first try to match by ID, then by SKU.

Example 4: Mixed ID and SKU Usage

```
{
  "partnerOrderId": "PARTNER-20260117-004",
  "customer": {
    "firstName": "Sarah",
    "lastName": "Williams",
    "email": "sarah.w@example.com",
    "phone": "555-0196"
  },
  "shippingAddress": {
    "firstName": "Sarah",
    "lastName": "Williams",
    "address1": "500 Commerce Street",
    "city": "Chicago",
    "state": "IL",
    "zip": "60601",
    "country": "US"
  },
  "items": [
    {
      "variantId": "clxyz789ghi012jkl345",
      "quantity": 1
    },
    {
      "variantId": "COLLAGEN-50G",
      "quantity": 2
    }
  ]
}
```

Example 4: JavaScript/Node.js Implementation

```javascript
const axios = require('axios');

async function createOrder(orderData) {
  try {
    const response = await axios.post(
      'https://pepapi.stmin.dev/api/sales-channels/integration/orders',
      orderData,
      {
        headers: {
          'Content-Type': 'application/json',
          'X-API-Key': process.env.CENTRE_RESEARCH_API_KEY
        }
      }
    );

    console.log('Order created successfully:', response.data);
    return response.data;
  } catch (error) {
    console.error('Failed to create order:', error.response?.data || error.message);
    throw error;
  }
}

// Usage
const orderData = {
  partnerOrderId: `PARTNER-${Date.now()}`,
  customer: {
    firstName: "John",
    lastName: "Doe",
    email: "john.doe@example.com",
    phone: "555-0199"
  },
  shippingAddress: {
    firstName: "John",
    lastName: "Doe",
    address1: "123 Main Street",
    city: "Los Angeles",
    state: "CA",
    zip: "90001",
    country: "US"
  },
  items: [
    {
      variantId: "clxyz123abc456def789",
      quantity: 2
    }
  ]
};

createOrder(orderData)
```

```
  .then(result => console.log('Success:', result))
  .catch(error => console.error('Error:', error));
```

Example 5: Python Implementation

```python
import requests
import os
from datetime import datetime

def create_order(order_data):
    url = "https://pepapi.stmin.dev/api/sales-channels/integration/orders"
    headers = {
        "Content-Type": "application/json",
        "X-API-Key": os.environ.get("CENTRE_RESEARCH_API_KEY")
    }

    try:
        response = requests.post(url, json=order_data, headers=headers)
        response.raise_for_status()
        print(f"Order created successfully: {response.json()}")
        return response.json()
    except requests.exceptions.HTTPError as e:
        print(f"Failed to create order: {e.response.text}")
        raise

# Usage
order_data = {
    "partnerOrderId": f"PARTNER-{int(datetime.now().timestamp())}",
    "customer": {
        "firstName": "John",
        "lastName": "Doe",
        "email": "john.doe@example.com",
        "phone": "555-0199"
    },
    "shippingAddress": {
        "firstName": "John",
        "lastName": "Doe",
        "address1": "123 Main Street",
        "city": "Los Angeles",
        "state": "CA",
        "zip": "90001",
        "country": "US"
    },
    "items": [
        {
            "variantId": "clxyz123abc456def789",
            "quantity": 2
        }
    ]
}
```

```
result = create_order(order_data)
```

Error Responses

401 Unauthorized - Missing API Key

Status Code: `401`

```
{
  "success": false,
  "error": "Missing API Key"
}
```

Cause: The `X-API-Key` header was not included in the request.

Solution: Include the API key in request headers.

---

401 Unauthorized - Invalid API Key

Status Code: `401`

```
{
  "success": false,
  "error": "Invalid API Key"
}
```

Cause: The provided API key does not match any active sales channel.

Solution: Verify your API key is correct and contact support if the issue persists.

---

403 Forbidden - Inactive Sales Channel

Status Code: `403`

```
{
  "success": false,
  "error": "Sales Channel is not active"
}
```

Cause: Your sales channel account has been paused or deactivated.

Solution: Contact Centre Research support to reactivate your account.

---

400 Bad Request - Duplicate Order

Status Code: `400`

```
{
  "success": false,
```

```
  "error": "An order with this partnerOrderId already exists for this channel.",
  "orderId": "clp123abc456def789ghi",
  "orderNumber": "ORD-1705329600-ABC123"
}
```

Cause: An order with the same `partnerOrderId` has already been created.

Solution: Use a unique `partnerOrderId` for each order. If this is intentional, use the returned order details instead of creating a duplicate.

---

400 Bad Request - Missing Items

Status Code: `400`

```
{
  "success": false,
  "error": "Items array is required"
}
```

Cause: The `items` array is missing or empty.

Solution: Include at least one item in the `items` array.

---

400 Bad Request - Invalid Quantity

Status Code: `400`

```
{
  "success": false,
  "error": "Invalid quantity for variant clxyz123abc456def789"
}
```

Cause: Quantity is less than 1 or not a valid number.

Solution: Ensure all quantities are positive integers (≥ 1).

---

400 Bad Request - Price Not Configured

Status Code: `400`

```
{
  "success": false,
  "error": "Price not configured for variant BPC-157-5MG on this channel. Tried both ID and SKU lookup."
}
```

Cause: The requested product variant does not have a price configured in your sales channel's price list. The system tried to find the variant by both database ID and SKU code.

Solution:

1. Verify the variant ID or SKU is correct
2. Contact Centre Research to add pricing for this product variant to your channel

---

400 Bad Request - Missing Required Fields

Status Code: `400`

```
{
  "success": false,
  "error": "Validation error",
  "details": [
    "customer.email is required",
    "shippingAddress.address1 is required"
  ]
}
```

Cause: One or more required fields are missing from the request.

Solution: Review the request format and ensure all required fields are included.

---

500 Internal Server Error

Status Code: `500`

```
{
  "success": false,
  "error": "Internal server error",
  "message": "An unexpected error occurred"
}
```

Cause: An unexpected error occurred on the server.

Solution: Retry the request with exponential backoff. If the issue persists, contact support with the request details.

Order Status Flow

After order creation, orders progress through the following statuses:

Status	Description
PENDING	Order received, awaiting processing
PROCESSING	Order is being prepared for shipment
LABEL_CREATED	Shipping label has been generated
SHIPPED	Order has been dispatched to carrier
DELIVERED	Order successfully delivered to customer
CANCELLED	Order was cancelled (may trigger refund)

REFUNDED	Payment refunded to customer
ON_HOLD	Order requires attention or has an issue

Status Transitions

```
PENDING → PROCESSING → LABEL_CREATED → SHIPPED → DELIVERED
   ↓           ↓              ↓            ↓
CANCELLED   CANCELLED      CANCELLED    CANCELLED
   ↓           ↓              ↓            ↓
REFUNDED    REFUNDED       REFUNDED     REFUNDED
```

Note: Orders can be placed on hold at any stage before delivery if issues are detected.

Rate Limiting

To ensure fair usage and system stability, the following rate limits apply:

Limit Type	Value
Per Minute	100 requests
Burst Window	200 requests per 10 seconds
Concurrent Requests	10 simultaneous connections

Rate Limit Headers

Response headers include rate limit information:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 95
X-RateLimit-Reset: 1705329660
```

429 Too Many Requests

Status Code: `429`

```
{
  "success": false,
  "error": "Rate limit exceeded",
  "retryAfter": 60
}
```

Solution: Implement exponential backoff and respect the `retryAfter` value (in seconds).

Best Practices

1. Generate Unique Partner Order IDs

Always use a unique identifier from your system as the `partnerOrderId` to prevent duplicate orders.

Good:

```
partnerOrderId: `PARTNER-${Date.now()}-${yourInternalOrderId}`
```

Bad:

```
partnerOrderId: "order-123" // May conflict with existing orders
```

2. Implement Retry Logic

Implement retry logic with exponential backoff for transient failures:

```
async function createOrderWithRetry(orderData, maxRetries = 3) {
  for (let i = 0; i < maxRetries; i++) {
    try {
      const response = await fetch(
        'https://pepapi.stmin.dev/api/sales-channels/integration/orders',
        {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'X-API-Key': process.env.API_KEY
          },
          body: JSON.stringify(orderData)
        }
      );

      if (response.ok) {
        return await response.json();
      }

      // Don't retry client errors (4xx)
      if (response.status >= 400 && response.status < 500) {
        throw new Error(await response.text());
      }

      // Retry server errors (5xx) with exponential backoff
      const delay = Math.pow(2, i) * 1000;
      await new Promise(resolve => setTimeout(resolve, delay));
    } catch (error) {
      if (i === maxRetries - 1) throw error;
    }
  }
}
```

3. Validate Data Before Sending

Perform client-side validation to catch errors early:

```
function validateOrder(order) {
  // Required fields
  if (!order.partnerOrderId) {
    throw new Error('partnerOrderId is required');
  }

  if (!order.items || order.items.length === 0) {
    throw new Error('At least one item is required');
  }

  // Validate items
  for (const item of order.items) {
    if (!item.variantId) {
      throw new Error('variantId is required for all items');
    }
    if (!item.quantity || item.quantity < 1) {
      throw new Error('Quantity must be at least 1');
    }
  }

  // Validate customer
  if (!order.customer?.email) {
    throw new Error('Customer email is required');
  }

  // Validate email format
  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(order.customer.email)) {
    throw new Error('Invalid email format');
  }

  // Validate shipping address
  const requiredAddressFields = ['firstName', 'lastName', 'address1', 'city',
'state', 'zip'];
  for (const field of requiredAddressFields) {
    if (!order.shippingAddress?.[field]) {
      throw new Error(`shippingAddress.${field} is required`);
    }
  }

  // Validate state code (2 letters)
  if (!/^[A-Z]{2}$/.test(order.shippingAddress.state)) {
    throw new Error('State must be a 2-letter code (e.g., CA, NY)');
  }
}
```

4. Store Order Mapping

Maintain a mapping between your order IDs and Centre Research order IDs:

```
const orderMapping = new Map();

async function createAndMapOrder(yourOrderId, orderData) {
  orderData.partnerOrderId = `PARTNER-${yourOrderId}`;

  const response = await createOrder(orderData);

  // Store mapping
  orderMapping.set(yourOrderId, {
    centreResearchOrderId: response.data.orderId,
    centreResearchOrderNumber: response.data.orderNumber,
    status: response.data.status,
    createdAt: new Date().toISOString()
  });

  // Persist to your database
  await saveOrderMapping(yourOrderId, orderMapping.get(yourOrderId));

  return response;
}
```

5. Monitor API Health

Implement monitoring to track API performance and errors:

```
class APIMonitor {
  constructor() {
    this.metrics = {
      totalRequests: 0,
      successfulRequests: 0,
      failedRequests: 0,
      averageResponseTime: 0,
      errorsByType: {}
    };
  }

  async trackRequest(apiCall) {
    const startTime = Date.now();
    this.metrics.totalRequests++;

    try {
      const result = await apiCall();
      this.metrics.successfulRequests++;
      return result;
    } catch (error) {
      this.metrics.failedRequests++;

      // Track error types
      const errorType = error.response?.status || 'network_error';
      this.metrics.errorsByType[errorType] =
        (this.metrics.errorsByType[errorType] || 0) + 1;
```

```
      throw error;
    } finally {
      const duration = Date.now() - startTime;
      this.metrics.averageResponseTime =
        (this.metrics.averageResponseTime * (this.metrics.totalRequests - 1) +
duration)
        / this.metrics.totalRequests;
    }
  }

  getMetrics() {
    return {
      ...this.metrics,
      successRate: (this.metrics.successfulRequests / this.metrics.totalRequests *
100).toFixed(2) + '%'
    };
  }
}
```

6. Handle Idempotency

Detect and handle duplicate order scenarios gracefully:

```
async function createOrderSafely(orderData) {
  try {
    return await createOrder(orderData);
  } catch (error) {
    // Check if it's a duplicate order error
    if (error.response?.status === 400 &&
        error.response?.data?.error?.includes('partnerOrderId already exists')) {
      console.log('Order already exists:', error.response.data);

      // Return the existing order details instead of failing
      return {
        success: true,
        data: {
          orderId: error.response.data.orderId,
          orderNumber: error.response.data.orderNumber,
          status: 'EXISTING'
        },
        isDuplicate: true
      };
    }

    // Re-throw other errors
    throw error;
  }
}
```

7. Secure API Key Management

Never expose your API key in client-side code or version control:

```
// ✅ Good - Using environment variables
const API_KEY = process.env.CENTRE_RESEARCH_API_KEY;

// ❌ Bad - Hardcoded API key
const API_KEY = "your-actual-api-key-here";

// ✅ Good - Loading from secure configuration
const config = require('./config/secure-config');
const API_KEY = config.centreResearch.apiKey;
```

Testing

Test Environment

Base URL: Contact Centre Research for test environment access

Test Credentials

Contact Centre Research support to obtain:

- Test API key
- Test product variant IDs
- Test sales channel configuration

Sample Test Data

Use the following test data for integration testing:

```
{
  "partnerOrderId": "TEST-001",
  "customer": {
    "firstName": "Test",
    "lastName": "Customer",
    "email": "test@example.com",
    "phone": "555-0100"
  },
  "shippingAddress": {
    "firstName": "Test",
    "lastName": "Customer",
    "address1": "123 Test Street",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90001",
    "country": "US"
  },
  "items": [
    {
      "variantId": "test-variant-id",
```

```
      "quantity": 1
    }
  ]
}
```

Testing Checklist

- [ ] Verify API key authentication works
- [ ] Test successful order creation
- [ ] Test duplicate order prevention (same partnerOrderId)
- [ ] Test missing required fields validation
- [ ] Test invalid data validation (negative quantities, invalid email, etc.)
- [ ] Test with single and multiple items
- [ ] Test with and without billing address
- [ ] Test error handling and retries
- [ ] Test rate limiting behavior
- [ ] Verify order mapping and tracking

Support

Technical Support

Email: api-support@centreresearch.org
Response Time: Within 24 hours (business days)
Emergency Support: Contact your dedicated account manager

Documentation

This documentation is maintained by Centre Research.
Last Updated: January 17, 2026
Version: 1.0.0

Feedback

We welcome your feedback to improve this documentation and our API:
Email: documentation@centreresearch.org

Common Issues

Issue	Solution
API key not working	Verify key is active and matches your sales channel
Price not configured error	Contact support to add products to your price list
Rate limit exceeded	Implement retry logic with exponential backoff
Address validation fails	Ensure state codes are 2-letter format (CA, NY, etc.)
Duplicate order error	Use unique partnerOrderId for each order

---

Appendix A: US State Codes

State	Code	State	Code
Alabama	AL	Montana	MT
Alaska	AK	Nebraska	NE
Arizona	AZ	Nevada	NV
Arkansas	AR	New Hampshire	NH
California	CA	New Jersey	NJ
Colorado	CO	New Mexico	NM
Connecticut	CT	New York	NY
Delaware	DE	North Carolina	NC
Florida	FL	North Dakota	ND
Georgia	GA	Ohio	OH
Hawaii	HI	Oklahoma	OK
Idaho	ID	Oregon	OR
Illinois	IL	Pennsylvania	PA
Indiana	IN	Rhode Island	RI
Iowa	IA	South Carolina	SC
Kansas	KS	South Dakota	SD
Kentucky	KY	Tennessee	TN
Louisiana	LA	Texas	TX
Maine	ME	Utah	UT
Maryland	MD	Vermont	VT
Massachusetts	MA	Virginia	VA
Michigan	MI	Washington	WA
Minnesota	MN	West Virginia	WV
Mississippi	MS	Wisconsin	WI
Missouri	MO	Wyoming	WY

Full reference: https://www.usps.com/ship/official-abbreviations.htm

---

Appendix B: Integration Workflow

Recommended Integration Steps

1. **Initial Setup**
   - Receive API key from Centre Research
   - Obtain product variant IDs for your catalog
   - Configure price list with Centre Research team
   - Set up test environment
2. **Development**
   - Implement API client with authentication
   - Add order creation logic
   - Implement error handling and retries
   - Add validation for all required fields
   - Set up order tracking and mapping
3. **Testing**
   - Test with sample data in test environment
   - Verify error handling for all scenarios
   - Test rate limiting behavior
   - Validate idempotency (duplicate prevention)
   - Perform load testing
4. **Go Live**
   - Switch to production API URL
   - Update API key to production key
   - Monitor initial orders closely
   - Set up alerting for errors
   - Document your integration
5. **Ongoing**
   - Monitor API performance metrics
   - Handle error notifications
   - Keep product variant list updated
   - Coordinate price list changes with Centre Research

Appendix C: Quick Reference

Essential URLs

Purpose	URL
API Base URL	https://pepapi.stmin.dev/api
Create Order	https://pepapi.stmin.dev/api/sales-channels/integration/orders

Essential Headers

```
Content-Type: application/json
X-API-Key: your-api-key-here
```

Minimum Valid Request

```
{
  "partnerOrderId": "UNIQUE-ID",
  "customer": {
    "firstName": "John",
    "lastName": "Doe",
    "email": "john@example.com"
  },
  "shippingAddress": {
    "firstName": "John",
    "lastName": "Doe",
    "address1": "123 Main St",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90001",
    "country": "US"
  },
  "items": [
    {
      "variantId": "variant-id-here",
      "quantity": 1
    }
  ]
}
```

Changelog

Version 1.0.0 (January 17, 2026)

- Initial release of Sales Channel Integration API
- External order creation endpoint
- API key authentication
- Idempotency support with `partnerOrderId`
- Automatic customer management
- Channel-based pricing
- Comprehensive error handling
