E-Commerce Product API Documentation

Table of Contents

Overview

The E-Commerce Product API provides developers with a simple yet powerful interface to manage product information across e-commerce platforms. This RESTful API enables seamless integration with your existing systems, allowing you to programmatically access and manipulate product data with minimal development effort.

Key Features

- **Comprehensive Product Management**: Access, create, update, and delete product information through intuitive endpoints
- **Robust Categorization System**: Organize products with a flexible category structure
- **Detailed Inventory Tracking**: Monitor product availability with advanced inventory features
- **Powerful Search & Filtering**: Find specific products based on various parameters
- **Efficient Media Management**: Handle product images and other associated media
- **Secure Authentication**: Protect your data with industry-standard token-based authentication
- **Detailed Error Reporting**: Troubleshoot issues quickly with informative error messages
- **Compliant Data Structures**: All responses follow consistent JSON formatting

返回顶部

Business Benefits

- **Streamline Operations**: Automate product management tasks to reduce manual work
- **Improve Time-to-Market**: Rapidly update product information across multiple channels
- **Enhance Customer Experience**: Ensure product data consistency across all customer touchpoints
- **Reduce Integration Costs**: Standardized API reduces development and maintenance expenses
- **Scale Efficiently**: API designed to handle everything from small catalogs to enterprise-level inventories

返回顶部

Use Cases

- **Multi-Channel Commerce**: Synchronize product data across web stores, marketplaces, and physical locations
- **Custom Storefronts**: Build unique shopping experiences with real-time product information
- **Inventory Management Systems**: Create specialized tools for inventory control and optimization
- **Pricing Automation**: Implement dynamic pricing based on market conditions or inventory levels
- **Product Information Management (PIM)**: Centralize and distribute product data across your organization
- **Marketing Automation**: Connect product data with email campaigns and promotional activities

返回顶部

Authentication

All API requests require secure authentication using Bearer Tokens. This authentication method is industry-standard and provides a secure way to access the API while maintaining flexibility.

```
Authorization: Bearer YOUR_API_KEY
```

Authentication Flow

The diagram below illustrates the complete authentication process, including token refresh when needed:

```
sequenceDiagram
    participant Client
    participant Auth Server
    participant API Server

    Client->>Auth Server: 1. Request API key (POST /api/v1/auth/token)
    Auth Server-->>Client: 2. Return API key and refresh token

    Client->>API Server: 3. Send request with API key (Authorization: Bearer {token})
    API Server-->>Client: 4. Return requested resource

    Note over Client,API Server: API key expires

    Client->>API Server: 5. Send request with expired API key
    API Server-->>Client: 6. Return 401 error (token_expired)

    Client->>Auth Server: 7. Request new API key using refresh token (POST /api/v1/auth/refresh)
    Auth Server-->>Client: 8. Return new API key

    Client->>API Server: 9. Send request with new API key
    API Server-->>Client: 10. Return requested resource
```

返回顶部

Authentication Guide

1. Obtaining an API Key

To access the API, you first need to obtain an API key through the authentication endpoint:

Request Example:

```
POST /api/v1/auth/token
Content-Type: application/json

{
  "client_id": "your_client_id",
  "client_secret": "your_client_secret",
  "grant_type": "client_credentials"
}
```

Response Example:

```
{
  "status": "success",
  "data": {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "Bearer",
    "expires_in": 3600
  },
  "meta": {
    "request_id": "req_abc123",
    "timestamp": "2023-06-20T14:15:00Z"
  }
}
```

2. Using Your API Key

Include the obtained API key in the Authorization header of all subsequent requests:

```
GET /api/v1/products
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

3. Handling Token Expiration

API keys typically expire after a certain period. When a key expires, you need to request a new one using your refresh token:

Request Example:

```
POST /api/v1/auth/refresh
Content-Type: application/json

{
  "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "grant_type": "refresh_token"
}
```

Response Example:

```
{
  "status": "success",
  "data": {
    "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "token_type": "Bearer",
    "expires_in": 3600
  },
  "meta": {
    "request_id": "req_def456",
    "timestamp": "2023-06-20T15:30:00Z"
  }
}
```

返回顶部

Token Management Examples

For efficient token management, we recommend implementing a token manager in your application. Below are examples in different programming languages:

JavaScript/Node.js

```
// Token Management Class
class TokenManager {
  constructor() {
    this.accessToken = null;
    this.refreshToken = null;
    this.expiresAt = null;
  }

  // Set tokens
  setTokens(accessToken, refreshToken, expiresIn) {
    this.accessToken = accessToken;
    this.refreshToken = refreshToken;
    // Set expiration time (expire 5 minutes early for safety)
    this.expiresAt = Date.now() + (expiresIn - 300) * 1000;
  }

  // Get valid token
  async getValidToken() {
    // If token doesn't exist or has expired, refresh it
    if (!this.accessToken || this.isTokenExpired()) {
      await this.refreshAccessToken();
    }
    return this.accessToken;
  }

  // Check if token is expired
  isTokenExpired() {
    return Date.now() >= this.expiresAt;
  }

  // Refresh access token
  async refreshAccessToken() {
    try {
      const response = await axios.post('https://api.example.com/api/v1/auth/refresh', {
        refresh_token: this.refreshToken,
        grant_type: 'refresh_token'
      });

      const { access_token, refresh_token, expires_in } = response.data.data;
      this.setTokens(access_token, refresh_token, expires_in);
    } catch (error) {
      console.error('Failed to refresh token:', error);
      // If refresh fails, user may need to log in again
      throw new Error('Authentication failed, please log in again');
    }
  }
}

// API client using token manager
class ApiClient {
  constructor(baseUrl, clientId, clientSecret) {
    this.baseUrl = baseUrl;
    this.clientId = clientId;
    this.clientSecret = clientSecret;
    this.tokenManager = new TokenManager();
  }
```

```javascript
    // Initialize (get initial token)
    async initialize() {
      try {
        const response = await axios.post(`${this.baseUrl}/api/v1/auth/token`, {
          client_id: this.clientId,
          client_secret: this.clientSecret,
          grant_type: 'client_credentials'
        });

        const { access_token, refresh_token, expires_in } = response.data.data;
        this.tokenManager.setTokens(access_token, refresh_token, expires_in);
      } catch (error) {
        console.error('Failed to initialize API client:', error);
        throw error;
      }
    }

    // Send API request
    async request(method, endpoint, data = null) {
      try {
        // Get valid token
        const token = await this.tokenManager.getValidToken();

        // Send request
        const response = await axios({
          method,
          url: `${this.baseUrl}${endpoint}`,
          data,
          headers: {
            'Authorization': `Bearer ${token}`,
            'Content-Type': 'application/json'
          }
        });

        return response.data;
      } catch (error) {
        if (error.response && error.response.status === 401) {
          // If authentication error, try to refresh token and retry
          await this.tokenManager.refreshAccessToken();
          return this.request(method, endpoint, data);
        }
        throw error;
      }
    }

    // Convenience methods
    async get(endpoint) {
      return this.request('GET', endpoint);
    }

    async post(endpoint, data) {
      return this.request('POST', endpoint, data);
    }

    async put(endpoint, data) {
      return this.request('PUT', endpoint, data);
    }

    async delete(endpoint) {
      return this.request('DELETE', endpoint);
    }
  }

  // Usage example
  async function example() {
    const apiClient = new ApiClient(
      'https://api.example.com',
      'your_client_id',
      'your_client_secret'
    );

    // Initialize client
    await apiClient.initialize();
    // Use client to send requests
    try {
      const products = await apiClient.get('/api/v1/products');
      console.log('Product list:', products);

      const newProduct = await apiClient.post('/api/v1/products', {
        name: 'New Product',
        description: 'Product description',
        price: 99.99,
        category_id: 'electronics'
      });
      console.log('Created product:', newProduct);
    } catch (error) {
      console.error('API request failed:', error);
    }
  }
```

返回顶部

API Endpoints

Product Endpoints

Get All Products

```
GET /api/v1/products
```

Parameter	Data Type	Required	Default	Description
page	Integer	No	1	Page number for pagination
limit	Integer	No	20	Results per page, max 100
category_id	String	No	-	Filter products by category ID
status	String	No	-	Filter by product status (active, draft, archived)
sort	String	No	created_at	Sort field (created_at, updated_at, price, name)
order	String	No	desc	Sort order (asc, desc)
search	String	No	-	Search in product name or description

Get a Single Product

```
GET /api/v1/products/{product_id}
```

Parameter	Data Type	Required	Default	Description
product_id	String	Yes	-	Product ID

Create a Product

```
POST /api/v1/products
```

Parameter	Data Type	Required	Default	Description
name	String	Yes	-	Product name
description	String	Yes	-	Product description
price	Number	Yes	-	Product price
currency	String	No	USD	Currency code
category_id	String	Yes	-	Product category ID
inventory	Integer	No	0	Inventory quantity
images	Array	No	[]	Array of product image URLs
attributes	Object	No	{}	Product attributes (color, size, etc.)
status	String	No	active	Product status (active, draft, archived)

Update a Product

```
PUT /api/v1/products/{product_id}
```

Parameter	Data Type	Required	Default	Description
product_id	String	Yes	-	Product ID
name	String	No	-	Product name
description	String	No	-	Product description
price	Number	No	-	Product price
currency	String	No	-	Currency code
category_id	String	No	-	Product category ID
inventory	Integer	No	-	Inventory quantity
images	Array	No	-	Array of product image URLs
attributes	Object	No	-	Product attributes (color, size, etc.)
status	String	No	-	Product status (active, draft, archived)

Delete a Product

```
DELETE /api/v1/products/{product_id}
```

Parameter	Data Type	Required	Default	Description

Parameter	Data Type	Required	Default	Description
product_id	String	Yes	-	Product ID

返回顶部

Category Endpoints

Get All Categories

```
GET /api/v1/categories
```

Parameter	Data Type	Required	Default	Description
page	Integer	No	1	Page number for pagination
limit	Integer	No	20	Results per page, max 100
parent_id	String	No	-	Filter by parent category ID

Get a Single Category

```
GET /api/v1/categories/{category_id}
```

Parameter	Data Type	Required	Default	Description
category_id	String	Yes	-	Category ID

返回顶部

Response Examples

Success Responses

Get Products List Response

```
{
  "status": "success",
  "data": {
    "items": [
      {
        "id": "prod_123456",
        "name": "Wireless Headphones",
        "description": "Premium noise-canceling wireless headphones with Bluetooth 5.0",
        "price": 99.99,
        "currency": "USD",
        "inventory": {
          "available": 157,
          "reserved": 3,
          "total": 160
        },
        "category": {
          "id": "electronics",
          "name": "Electronics",
          "parent_id": null
        },
        "images": [
          {
            "url": "https://example.com/images/headphones-main.jpg",
            "alt": "Wireless headphones front view",
            "is_primary": true
          },
          {
            "url": "https://example.com/images/headphones-angle.jpg",
            "alt": "Wireless headphones side view",
            "is_primary": false
          }
        ],
        "attributes": {
          "color": "Black",
          "weight": "250g",
          "battery_life": "20 hours"
        },
        "status": "active",
        "created_at": "2023-06-15T10:30:00Z",
        "updated_at": "2023-06-20T14:15:00Z"
      }
    ],
    "pagination": {
      "total": 42,
      "page": 1,
      "limit": 20,
      "pages": 3,
      "has_next": true,
      "has_prev": false
    }
  },
  "meta": {
    "request_id": "req_abc123",
    "timestamp": "2023-06-20T14:15:00Z"
  }
}
```

Get Single Product Response

```
{
  "status": "success",
  "data": {
    "item": {
      "id": "prod_123456",
      "name": "Wireless Headphones",
      "description": "Premium noise-canceling wireless headphones with Bluetooth 5.0",
      "price": 99.99,
      "currency": "USD",
      "inventory": {
        "available": 157,
        "reserved": 3,
        "total": 160
      },
      "category": {
        "id": "electronics",
        "name": "Electronics",
        "parent_id": null
      },
      "images": [
        {
          "url": "https://example.com/images/headphones-main.jpg",
          "alt": "Wireless headphones front view",
          "is_primary": true
        },
        {
          "url": "https://example.com/images/headphones-angle.jpg",
          "alt": "Wireless headphones side view",
          "is_primary": false
        }
      ],
      "attributes": {
        "color": "Black",
        "weight": "250g",
        "battery_life": "20 hours"
      },
      "status": "active",
      "created_at": "2023-06-15T10:30:00Z",
      "updated_at": "2023-06-20T14:15:00Z"
    }
  },
  "meta": {
    "request_id": "req_abc123",
    "timestamp": "2023-06-20T14:15:00Z"
  }
}
```

Create Product Response

```
{
  "status": "success",
  "data": {
    "item": {
      "id": "prod_789012",
      "name": "Smart Watch",
      "description": "Fitness and health tracking smart watch with heart rate monitor",
      "price": 149.99,
      "currency": "USD",
      "inventory": {
        "available": 50,
        "reserved": 0,
        "total": 50
      },
      "category": {
        "id": "electronics",
        "name": "Electronics",
        "parent_id": null
      },
      "images": [
        {
          "url": "https://example.com/images/smartwatch-main.jpg",
          "alt": "Smart watch front view",
          "is_primary": true
        },
        {
          "url": "https://example.com/images/smartwatch-side.jpg",
          "alt": "Smart watch side view",
          "is_primary": false
        }
      ],
      "attributes": {
        "color": "Silver",
        "weight": "45g",
        "battery_life": "5 days",
        "water_resistant": "IP68"
      },
      "status": "active",
      "created_at": "2023-07-05T09:45:00Z",
      "updated_at": "2023-07-05T09:45:00Z"
    }
  },
  "meta": {
    "request_id": "req_def456",
    "timestamp": "2023-07-05T09:45:00Z"
  }
}
```

返回顶部

Error Responses

Validation Error

```json
{
  "status": "error",
  "error": {
    "code": "validation_error",
    "message": "The provided product data is invalid",
    "details": [
      {
        "field": "price",
        "message": "Price must be greater than zero"
      },
      {
        "field": "name",
        "message": "Product name is required"
      },
      {
        "field": "category_id",
        "message": "Category ID does not exist"
      }
    ]
  },
  "meta": {
    "request_id": "req_ghi789",
    "timestamp": "2023-07-05T10:15:00Z"
  }
}
```

Authentication Error

```json
{
  "status": "error",
  "error": {
    "code": "unauthorized",
    "message": "Authentication failed",
    "details": "The provided API key is invalid or expired"
  },
  "meta": {
    "request_id": "req_jkl012",
    "timestamp": "2023-07-05T10:20:00Z"
  }
}
```

Resource Not Found Error

```json
{
  "status": "error",
  "error": {
    "code": "not_found",
    "message": "Resource not found",
    "details": "Product with ID 'prod_999999' does not exist"
  },
  "meta": {
    "request_id": "req_mno345",
    "timestamp": "2023-07-05T10:25:00Z"
  }
}
```

Rate Limit Error

```json
{
  "status": "error",
  "error": {
    "code": "rate_limit_exceeded",
    "message": "Rate limit exceeded",
    "details": "Maximum of 100 requests per minute allowed, please try again later"
  },
  "meta": {
    "request_id": "req_pqr678",
    "timestamp": "2023-07-05T10:30:00Z",
    "rate_limit": {
      "limit": 100,
      "remaining": 0,
      "reset": 1625097600
    }
  }
}
```

返回顶部

Code Examples

JavaScript/Node.js

```javascript
// Using axios for API calls
const axios = require('axios');

const API_KEY = 'your_api_key';
const BASE_URL = 'https://api.example.com';

// Configure axios instance
const api = axios.create({
  baseURL: BASE_URL,
  headers: {
    'Authorization': `Bearer ${API_KEY}`,
    'Content-Type': 'application/json'
  }
});

// Get all products
async function getAllProducts(page = 1, limit = 20) {
  try {
    const response = await api.get('/api/v1/products', {
      params: { page, limit }
    });
    return response.data;
  } catch (error) {
    handleError(error);
  }
}

// Create new product
async function createProduct(productData) {
  try {
    const response = await api.post('/api/v1/products', productData);
    return response.data;
  } catch (error) {
    handleError(error);
  }
}

// Error handling
function handleError(error) {
  if (error.response) {
    console.error('API error:', error.response.data);
  } else if (error.request) {
    console.error('Network error:', error.request);
  } else {
    console.error('Error:', error.message);
  }
}
```

Python

```python
import requests
from typing import Dict, Optional

class EcommerceAPI:
    def __init__(self, api_key: str, base_url: str = "https://api.example.com"):
        self.base_url = base_url
        self.headers = {
            "Authorization": f"Bearer {api_key}",
            "Content-Type": "application/json"
        }

    def get_products(self, page: int = 1, limit: int = 20) -> Dict:
        """Get product list"""
        try:
            response = requests.get(
                f"{self.base_url}/api/v1/products",
                headers=self.headers,
                params={"page": page, "limit": limit}
            )
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            self._handle_error(e)

    def create_product(self, product_data: Dict) -> Dict:
        """Create new product"""
        try:
            response = requests.post(
                f"{self.base_url}/api/v1/products",
                headers=self.headers,
                json=product_data
            )
            response.raise_for_status()
            return response.json()
        except requests.exceptions.RequestException as e:
            self._handle_error(e)

    def _handle_error(self, error: Exception) -> None:
        """Error handling"""
        if isinstance(error, requests.exceptions.HTTPError):
            print(f"HTTP error: {error.response.json()}")
        elif isinstance(error, requests.exceptions.ConnectionError):
            print("Connection error: Cannot connect to server")
        else:
            print(f"Error: {str(error)}")
```

返回顶部

Error Handling

The API uses standard HTTP status codes to indicate the success or failure of requests:

- 200 OK: Request successful
- 201 Created: Resource successfully created
- 400 Bad Request: Invalid request
- 401 Unauthorized: Authentication failed

- 403 Forbidden: Insufficient permissions
- 404 Not Found: Resource not found
- 429 Too Many Requests: Rate limit exceeded
- 500 Internal Server Error: Server error

Error Codes Reference

Error Code	HTTP Status	Description	Possible Causes	Solutions
`validation_error`	400	Request parameter validation failed	Request contains invalid or missing required fields	Check that request parameters meet API requirements, ensure all required fields are provided and correctly formatted
`invalid_json`	400	Invalid JSON format	Request body is not valid JSON	Check JSON syntax, ensure all quotes, brackets, and commas are correctly matched
`invalid_product_data`	400	Product data is invalid	Product data doesn't comply with business rules	Check that product data complies with business rules, such as price must be greater than zero
`invalid_category_id`	400	Category ID is invalid	Provided category ID doesn't exist	Use a valid category ID, which can be found via the categories API
`invalid_image_url`	400	Image URL is invalid	Provided image URL is inaccessible or incorrectly formatted	Ensure image URLs are publicly accessible and correctly formatted
`unauthorized`	401	Authentication failed	API key is missing, invalid, or expired	Check that your API key is correct, regenerate it if necessary
`token_expired`	401	Token has expired	Authentication token used has expired	Get a new API key and update authentication information
`forbidden`	403	Insufficient permissions	Don't have permission to perform the requested operation	Check account permissions, contact admin if needed for permission elevation
`not_found`	404	Resource not found	Requested resource doesn't exist	Check if resource ID is correct, confirm resource hasn't been deleted
`rate_limit_exceeded`	429	Rate limit exceeded	Too many requests sent in a short time	Implement request throttling, use caching to reduce API calls, wait for rate limit to reset

Error Code	HTTP Status	Description	Possible Causes	Solutions
`internal_error`	500	Internal server error	Server encountered an error processing the request	Log error details and contact API support team
`service_unavailable`	503	Service unavailable	Server temporarily unable to handle request	Implement retry mechanism, try again later
`maintenance_mode`	503	Maintenance mode	Server is undergoing maintenance	Wait for maintenance to complete before retrying
`duplicate_product`	409	Product already exists	Attempting to create a product that already exists	Check if product already exists, or use update API instead of create
`inventory_error`	400	Inventory error	Inventory operation doesn't comply with business rules	Check that inventory operation complies with business rules, such as inventory can't be negative

返回顶部

Error Handling Best Practices

1. **Implement Global Error Handling**: Create a unified error handling mechanism in your client code to catch and appropriately process all potential errors.
2. **Use Retry Mechanisms**: For temporary errors (such as 429, 503), implement an exponential backoff retry strategy.
3. **Log Detailed Information**: Record error details including request ID, timestamp, and complete error response for debugging.
4. **Provide User-Friendly Messages**: Display friendly error messages to end users without exposing technical details.
5. **Monitor Error Rates**: Track API error rates to quickly identify and resolve issues.
6. **Implement Circuit Breaker Pattern**: Temporarily stop sending requests to the API when error rates exceed thresholds to prevent cascading failures.
7. **Validate Request Data**: Validate data before sending requests to reduce server-side validation errors.
8. **Handle Partial Success**: For batch operations, implement partial success handling to ensure partial failures don't affect the entire operation.

返回顶部

Rate Limits

To ensure fair usage and optimal performance for all users, the API implements rate limiting:

- Standard limit: 100 requests per minute
- Bulk operations count as multiple requests based on the number of items

Rate limit information is included in response headers:

```
X-RateLimit-Limit: 100
X-RateLimit-Remaining: 95
X-RateLimit-Reset: 1625097600
```

Rate Limit Best Practices

1. **Monitor Rate Limit Headers**: Track remaining requests to avoid hitting limits
2. **Implement Throttling**: Space out requests to stay under limits
3. **Use Bulk Operations**: Combine multiple operations in a single request when possible
4. **Implement Caching**: Cache frequently accessed data to reduce API calls
5. **Use Exponential Backoff**: When rate limited, wait progressively longer between retries

返回顶部

Best Practices

1. **Use Authentication Properly**

   - Store API keys securely, never expose them in client-side code
   - Implement proper token refresh mechanisms
   - Rotate secrets periodically for enhanced security

2. **Optimize Request Patterns**

   - Use pagination for large datasets
   - Only request the data you need
   - Batch related operations when possible

3. **Implement Proper Error Handling**

   - Handle all potential error codes
   - Retry transient failures with exponential backoff
   - Present user-friendly messages for API errors

4. **Optimize Performance**

   - Cache frequently accessed data
   - Implement request compression for large payloads
   - Use conditional requests (If-Modified-Since) when appropriate

5. **Ensure Data Consistency**

   - Validate data before sending to the API
   - Implement proper synchronization for distributed systems
   - Use transactions for related operations when available

6. **Follow Security Best Practices**

- Use HTTPS for all requests
- Implement proper input sanitization
- Follow the principle of least privilege for API access

返回顶部

Quick Start Guide

Follow these steps to quickly integrate with the E-Commerce Product API:

1. **Register for API Access**

    - Create an account at our Developer Portal
    - Generate your API credentials (client ID and client secret)

2. **Authenticate**

    - Obtain an access token using your credentials
    - Store the token and refresh token securely

3. **Make Your First API Call**

    - Use the token to request a list of products
    - Examine the response structure to understand data format

4. **Implement Error Handling**

    - Set up proper error catching and handling

返回顶部