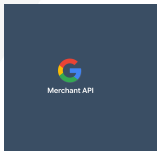
magmodules



Google Merchant API Integration

Google Shopping - Merchant API for Magento 2

Overview

This module integrates with the **Google Merchant API** (successor to Content API for Shopping) to enable real-time product synchronization with Google Merchant Center.

> **Note**: Content API for Shopping will be sunset on August 18, 2026. This module uses the new Merchant API.

API Documentation: https://developers.google.com/merchant/api

Merchant API Structure

The Merchant API is organized into multiple sub-APIs, each with its own PHP client library:

Sub-API	Package	Purpose
Products	`google/shopping-merchant-products`	Product management
Inventories	`google/shopping-merchant-inventories`	Local/regional inventory
Accounts	`google/shopping-merchant-accounts`	Account settings
Reports	`google/shopping-merchant-reports`	Performance data
Data Sources	`google/shopping-merchant-datasources`	Feed management

Requirements

- PHP 8.0 or higher
- Composer packages: `google/shopping-merchant-products`, `google/shopping-merchant-inventories`
- Google Cloud project with Merchant API enabled
- Service Account with Merchant Center access
- **API Data Source** created in Merchant Center (required for API uploads)

Installation

```
composer require google/shopping-merchant-products google/shopping-merchant-inventories
```

Authentication

Service Account (Recommended)

Service accounts are the preferred authentication method for server-to-server communication.

Setup Steps

1. Create a Google Cloud project at https://console.cloud.google.com
2. Enable the Merchant API
3. Create a Service Account with appropriate permissions
4. Download the JSON credentials file
5. Add the service account email to your Merchant Center account (Settings > Users)
6. Create an API Data Source in Merchant Center (Products > Feeds > Add data source > API)

Implementation

```
namespace Magmodules\GoogleShoppingApi\Service\Google;

use Google\Auth\Credentials\ServiceAccountCredentials;
use Google\Shopping\Merchant\Products\V1beta\Client\ProductInputsServiceClient;

class Authentication
{
    private const SCOPES = ['https://www.googleapis.com/auth/content'];

    public function __construct(
        private readonly ConfigRepository $config
    ) {}

    public function getCredentials(int $storeId): ServiceAccountCredentials
    {
        $jsonKey = $this->config->getServiceAccountJson($storeId);

        if (empty($jsonKey)) {
            throw new AuthenticationException('Service account credentials not configured');
        }

        return new ServiceAccountCredentials(
            self::SCOPES,
            json_decode($jsonKey, true)
```

```
        );
    }

    public function getProductInputsClient(int $storeId): ProductInputsServiceClient
    {
        return new ProductInputsServiceClient([
            'credentials' => $this->getCredentials($storeId)
        ]);
    }
}
```

API Resources

ProductInput Resource (Write Operations)

The `ProductInput` resource is used for all write operations (insert, update, delete). After processing, it becomes a `Product` resource.

Method	HTTP	Endpoint	Description
`insertProductInput`	POST	`/v1beta/accounts/{account}/productInputs`	Create/update product
`deleteProductInput`	DELETE	`/v1beta/accounts/{account}/productInputs/{id}`	Remove product
`updateProductInput`	PATCH	`/v1beta/accounts/{account}/productInputs/{id}`	Partial update

Product Resource (Read Operations)

The `Product` resource is read-only and represents the processed product.

Method	HTTP	Endpoint	Description
`getProduct`	GET	`/v1beta/accounts/{account}/products/{id}`	Get single product
`listProducts`	GET	`/v1beta/accounts/{account}/products`	List all products

Local Inventory Resource

For stores with local/physical inventory.

Method	HTTP	Endpoint	Desc

`insertLocalInventory`	POST	`/v1beta/accounts/{account}/products/{product}/localInventories`	Set lo inver
`deleteLocalInventory`	DELETE	`/v1beta/accounts/{account}/products/{product}/localInventories/{store}`	Remo inver
`listLocalInventories`	GET	`/v1beta/accounts/{account}/products/{product}/localInventories`	List inver

Regional Inventory Resource

For regional price/availability variations.

Method	HTTP	Endpoint
`insertRegionalInventory`	POST	`/v1beta/accounts/{account}/products/{product}/regionalInventories`
`deleteRegionalInventory`	DELETE	`/v1beta/accounts/{account}/products/{product}/regionalInventories/{region}`

Data Mapping

Product Identifier Format

Products are identified by: `{contentLanguage}~{feedLabel}~{offerId}`

Example: `en~US~SKU12345`

The full resource name format: `accounts/{account}/productInputs/{contentLanguage}~{feedLabel}~{offerId}`

Magento → Google Product Mapping

Google Field	Magento Source	Notes
`offerId`	SKU	Unique identifier, required
`contentLanguage`	Store locale	Two-letter ISO 639-1 code
`feedLabel`	Country code	Target country (e.g., "US")
`channel`	"ONLINE"	Online channel for e-commerce
`attributes.title`	Name	Max 150 chars

Google Field	Magento Source	Notes
`attributes.description`	Description	Max 5000 chars, HTML stripped
`attributes.link`	Product URL	Must be HTTPS
`attributes.imageLink`	Base image	Min 100x100px
`attributes.additionalImageLinks`	Gallery images	Up to 10 images
`attributes.price`	Regular price	Object: {amountMicros, currencyCode}
`attributes.salePrice`	Special price	Only if active
`attributes.availability`	Stock status	IN_STOCK / OUT_OF_STOCK / PREORDER
`attributes.brand`	Brand attribute	Required for most categories
`attributes.gtin`	EAN/UPC/ISBN	Strongly recommended
`attributes.mpn`	MPN attribute	Required if no GTIN
`attributes.condition`	Condition	new / refurbished / used
`attributes.googleProductCategory`	Mapped category	Google taxonomy ID

Price Format

The Merchant API uses micros for prices (1 USD = 1,000,000 micros):

```
use Google\Shopping\Merchant\Products\V1beta\Attributes;
use Google\Type\Price;

$price = new Price();
$price->setAmountMicros(1999000000); // $19.99
$price->setCurrencyCode('USD');

$attributes = new Attributes();
$attributes->setPrice($price);
```

Product Builder

```
namespace Magmodules\GoogleShoppingApi\Service\Google;
```

```
use Google\Shopping\Merchant\Products\V1beta\ProductInput;
use Google\Shopping\Merchant\Products\V1beta\Attributes;
use Google\Type\Price;

class ProductBuilder
{
    public function build(array $productData, string $feedLabel): ProductInput
    {
        $attributes = $this->buildAttributes($productData);

        $productInput = new ProductInput();
        $productInput->setOfferId($productData['id']);
        $productInput->setContentLanguage($productData['content_language'] ?? 'en');
        $productInput->setFeedLabel($feedLabel);
        $productInput->setChannel(\Google\Shopping\Merchant\Products\V1beta\Channel::ONLINE);
        $productInput->setAttributes($attributes);

        return $productInput;
    }

    private function buildAttributes(array $data): Attributes
    {
        $attributes = new Attributes();

        // Required fields
        $attributes->setTitle($this->truncate($data['title'] ?? '', 150));
        $attributes->setDescription($this->stripHtml($data['description'] ?? ''));
        $attributes->setLink($data['link'] ?? '');
        $attributes->setImageLink($data['image_link'] ?? '');
        $attributes->setAvailability($this->mapAvailability($data['availability'] ?? ''));

        // Price
        if (!empty($data['price'])) {
            $attributes->setPrice($this->buildPrice($data['price'], $data['currency'] ?? 'USD'));
        }

        // Sale price
        if (!empty($data['sale_price']) && $data['sale_price'] < $data['price']) {
            $attributes->setSalePrice($this->buildPrice($data['sale_price'], $data['currency'] ?? 'USD'));
        }

        // Identifiers
        if (!empty($data['brand'])) {
```

```
            $attributes->setBrand($data['brand']);
        }
        if (!empty($data['gtin'])) {
            $attributes->setGtin($data['gtin']);
        }
        if (!empty($data['mpn'])) {
            $attributes->setMpn($data['mpn']);
        }

        // Category
        if (!empty($data['google_product_category'])) {
            $attributes->setGoogleProductCategory($data['google_product_category']);
        }

        // Additional images
        if (!empty($data['additional_image_link'])) {
            $images = is_array($data['additional_image_link'])
                ? $data['additional_image_link']
                : [$data['additional_image_link']];
            $attributes->setAdditionalImageLinks(array_slice($images, 0, 10));
        }

        // Condition
        $attributes->setCondition($data['condition'] ?? 'new');

        return $attributes;
    }

    private function buildPrice(float $amount, string $currency): Price
    {
        $price = new Price();
        $price->setAmountMicros((int)($amount * 1000000));
        $price->setCurrencyCode($currency);
        return $price;
    }

    private function mapAvailability(string $availability): string
    {
        return match ($availability) {
            'in_stock', 'IN_STOCK' => 'IN_STOCK',
            'out_of_stock', 'OUT_OF_STOCK' => 'OUT_OF_STOCK',
            'preorder', 'PREORDER' => 'PREORDER',
            default => 'OUT_OF_STOCK',
```

```
        };
    }

    private function truncate(string $text, int $length): string
    {
        return mb_strlen($text) > $length ? mb_substr($text, 0, $length - 3) . '...' : $text;
    }

    private function stripHtml(string $text): string
    {
        return strip_tags(html_entity_decode($text));
    }
}
```

Product Adapter

```
namespace Magmodules\GoogleShoppingApi\Service\Google\Adapter;

use Google\Shopping\Merchant\Products\V1beta\Client\ProductInputsServiceClient;
use Google\Shopping\Merchant\Products\V1beta\Client\ProductsServiceClient;
use Google\Shopping\Merchant\Products\V1beta\InsertProductInputRequest;
use Google\Shopping\Merchant\Products\V1beta\DeleteProductInputRequest;
use Google\Shopping\Merchant\Products\V1beta\ProductInput;

class ProductAdapter
{
    public function __construct(
        private readonly Authentication $auth,
        private readonly ProductBuilder $productBuilder,
        private readonly ConfigRepository $config
    ) {}

    /**
     * Insert or update a product
     */
    public function insert(int $storeId, array $productData): ProductInput
    {
        $client = $this->auth->getProductInputsClient($storeId);
        $merchantId = $this->config->getMerchantId($storeId);
        $dataSource = $this->config->getDataSource($storeId);
        $feedLabel = $this->config->getFeedLabel($storeId);
```

```
        $productInput = $this->productBuilder->build($productData, $feedLabel);

        $request = new InsertProductInputRequest();
        $request->setParent(sprintf('accounts/%s', $merchantId));
        $request->setProductInput($productInput);
        $request->setDataSource($dataSource);

        return $client->insertProductInput($request);
    }

    /**
     * Delete a product
     */
    public function delete(int $storeId, string $offerId): void
    {
        $client = $this->auth->getProductInputsClient($storeId);
        $merchantId = $this->config->getMerchantId($storeId);
        $feedLabel = $this->config->getFeedLabel($storeId);
        $language = $this->config->getContentLanguage($storeId);
        $dataSource = $this->config->getDataSource($storeId);

        $name = sprintf(
            'accounts/%s/productInputs/%s~%s~%s',
            $merchantId,
            $language,
            $feedLabel,
            $offerId
        );

        $request = new DeleteProductInputRequest();
        $request->setName($name);
        $request->setDataSource($dataSource);

        $client->deleteProductInput($request);
    }

    /**
     * Build Google Product ID from product data
     */
    public function buildProductId(int $storeId, string $offerId): string
    {
        $feedLabel = $this->config->getFeedLabel($storeId);
```

```
        $language = $this->config->getContentLanguage($storeId);

        return sprintf('%s~%s~%s', $language, $feedLabel, $offerId);
    }
}
```

Rate Limits & Best Practices

Rate Limits

Resource	Limit
Overall API	Varies by account tier
Insert/Update	Process in batches
Processing time	Several minutes after submission

Best Practices

1. **Create an API Data Source** - Required before using the API
2. **Use batch operations** for bulk updates when available
3. **Implement exponential backoff** for rate limit errors
4. **Cache processed product IDs** to avoid unnecessary operations
5. **Handle processing delay** - Products may take minutes to appear after insert
6. **Use regional inventory** for price/availability variations by region

Error Handling

```
use Google\ApiCore\ApiException;

public function executeWithRetry(callable $operation, int $maxAttempts = 3): mixed
{
    $attempt = 0;
    $lastException = null;

    while ($attempt < $maxAttempts) {
        try {
            return $operation();
        } catch (ApiException $e) {
```

```
            $lastException = $e;
            $statusCode = $e->getCode();

            // Rate limit - exponential backoff
            if ($statusCode === 429) {
                $waitTime = (int)pow(2, $attempt) * 1000000; // microseconds
                usleep($waitTime);
                $attempt++;
                continue;
            }

            // Server error - retry
            if ($statusCode >= 500) {
                $attempt++;
                sleep(1);
                continue;
            }

            // Client error - don't retry
            throw $e;
        }
    }

    throw $lastException;
}
```

Testing Connection

```
namespace Magmodules\GoogleShoppingApi\Service\Google;

use Google\Shopping\Merchant\Products\V1beta\Client\ProductsServiceClient;
use Google\Shopping\Merchant\Products\V1beta\ListProductsRequest;
use Google\ApiCore\ApiException;

class ConnectionTest
{
    public function __construct(
        private readonly Authentication $auth,
        private readonly ConfigRepository $config
    ) {}
```

```
    public function test(int $storeId): array
    {
        try {
            $client = new ProductsServiceClient([
                'credentials' => $this->auth->getCredentials($storeId)
            ]);

            $merchantId = $this->config->getMerchantId($storeId);

            $request = new ListProductsRequest();
            $request->setParent(sprintf('accounts/%s', $merchantId));
            $request->setPageSize(1);

            $response = $client->listProducts($request);
            $products = iterator_to_array($response->iterateAllElements());

            return [
                'success' => true,
                'message' => 'Successfully connected to Google Merchant Center',
                'merchant_id' => $merchantId,
                'product_count' => count($products)
            ];
        } catch (ApiException $e) {
            return [
                'success' => false,
                'message' => $e->getMessage(),
                'code' => $e->getCode()
            ];
        }
    }
}
```

Migration from Content API

Key differences from Content API for Shopping:

Aspect	Content API	Merchant API
Package	`google/apiclient-services`	`google/shopping-merchant-*`
Service	`ShoppingContent`	Multiple specialized clients
Write resource	`Product`	`ProductInput`

Aspect	Content API	Merchant API
Read resource	`Product`	`Product`
Batch endpoint	`products.custombatch`	Individual requests
Inventory	`inventory.set`	`localInventories`, `regionalInventories`
Data source	Not required	Required (API type)
Product ID	`online:lang:country:sku`	`lang~feedLabel~sku`

References

- Merchant API Overview
- PHP Client Libraries
- Products API Reference
- Inventories API Reference
- Migration Guide

For a complete overview of features and configuration options, see the Google Shopping - Merchant API extension on magmodules.eu

All articles for Google Shopping - Merchant API

Installation

1	Installation using Composer (recommended)

Configuration

1	Google Cloud Service Account Setup

Other

1	**Google Merchant API Integration (current)**

Background

1	Google Shopping Merchant API Sync Explained
2	Using Feed and API Together

