Introduction

Welcome to the latest edition of our API documentation! Our API provides a way for our dealers to communicate with our system by transmitting data over the internet.

Build your application on the new and improved **WPS API v4** platform

Simple

Follows standardized, HTTP-based, REST-compliant API methods commonly used to interact over the Internet; combined with painless, token-based authentication and easy-to-use syntax.

How it works →

Powerful

With new features like filters, includes, product variants, and *more* you have tremendous flexibility in how you query and consume our extensive repertoire of product information.

New features guide →

Highly-available

Built on the largest cloud provider infrastructure designed to handle dense peak periods and wild swings in traffic patterns without any degradation of performance slowing you down.

Explore our architecture →

This area aims to be a comprehensive guide to Version 4 of the WPS API. We'll cover topics such as the purpose of the API, signing up for access, obtaining an access token, determining what resources we offer, making your first request, and what to do if you have trouble getting the information you need.

So what is the WPS API, exactly?

API stands for Application Programming Interface. The WPS API is an easy-to-use, robust, and reliable way for dealers and/or other third parties to obtain product information that Western Power Sports has tirelessly acquired, generated, and organized. This information can be used to populate your database, save and organize for import into software for your specific purpose, or even consume in real-time dynamically on your website. Whether you are writing a small script that quickly spins through our entire product line and imports vital pieces of product information into your system, or you wish to completely automate your entire product info pipeline to the various sales channels you interact with, the WPS API can help.

This is all provided to the WPS dealer network for free and is accessible 24/7/365 by simply signing up for access.

Before you begin

If you're reading this documentation, chances are you are interested in obtaining data from us in a server-to-server automated fashion. If you are not, you may find

the Data Depot downloads area more useful as it allows you to download the data in a point-and-click fashion for the purposes of importing into your system.

The obligatory out-of-your-depth warning

The API is designed to be used by programmers who are familiar with the transmission of data via HTTP in a RESTful way in JSON format. If you do not understand what those technologies mean, you may need to hire a programmer or seek additional help from a third party.

Simply put, if you are a programmer, I.T. manager, and/or a technology professional seeking a way to acquire and transmit data to and from Western Power Sports, continue reading.

Helpful Hints

Throughout this guide there are a number of small-but-handy pieces of information that can make using the WPS API easier, more interesting, and less hazardous. Here's what to look out for.

Pro Tips help you get more from the WPS API

These are tips and tricks that will help you be a WPS API wizard!

Notes are handy pieces of information

These are for the extra tidbits sometimes necessary to understand the WPS API.

Warnings help you not blow things up

Be aware of these messages if you wish to avoid certain death.

You'll see this by a feature that hasn't been released

Some pieces of this website are for future enhancements of WPS API that are not yet released.

Support

It is our goal at WPS to create the tools and services you need to succeed in this competitive market. If you have questions about permissions, acceptable uses, and best practices please let us know. If you are experiencing technical difficulties, please consult the API Documentation first. Many times the answer will be there. But if you still need help, please view our Support Page, or send us an e-mail web_services@wps-inc.com and we'll be glad to help.

Authentication

Authentication is the process of determining whether someone or something is, in fact, who or what it is declared to be. To authenticate with our API, you must have already signed up for access and received your API access token.

TLDR;

For the impatient, here's how to make a basic request to our API.

```
~ $ curl -H "Authorization: Bearer Abc123XYZ"
http://api.wps-inc.com/items
```

To use any part of the WPS API, you'll need an API access token. The WPS API uses access tokens to associate requests to API resources with your account. In this quick example, you would replace `Abc123XYZ` with the access token you were issued.

Assuming you have completed the signup process and have been issued an access token, the request would return a list of Items.

Just an example

Keep in mind that there are many, many other ways to make HTTP requests besides `curl`. We'll go into more detail throughout the documentation but the cURL example is one that many programmers/developers/engineers are familiar with.

What is an API Access Token?

An access token is a unique identifier of an application requesting access to our service. When you sign up, we review your submissions and generate an API access token for your application to use when requesting our API service. We can then match the token you provide to the one we store on our side in order to authenticate. API access tokens are a replacement to sending some username/password combination over HTTP which is not as secure.

Warning

Remember to keep your access tokens secret; treat them just like passwords! They act on your behalf when interacting with the API. Don't hard-code them into your programs. Instead, opt to use them as environment variables.

Using your token

"Okay, I signed up, I was issued an API access token, now what?"

Access tokens are used in the `Authorization` header of your HTTP requests.

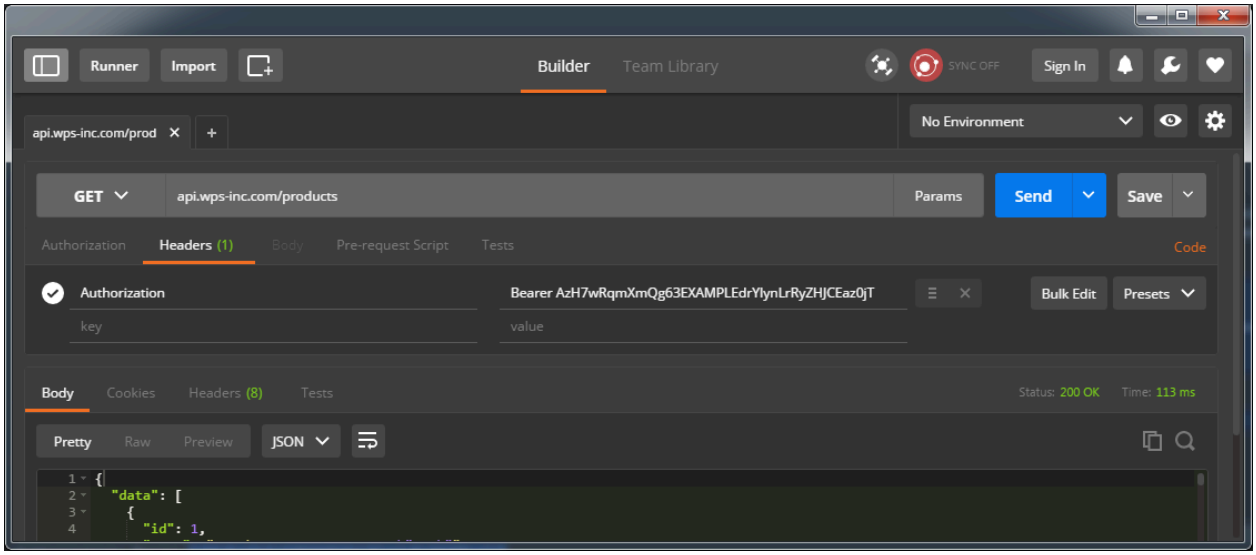
For example:

```
Authorization: Bearer AzH7wRqmXmQg63EXAMPLEdrYIynLrRyZHJCEaz0jT
```

Pro Tip→ Grab a good REST Client

A great way to try out our API and practice making HTTP requests is with a good REST client tool like Postman. Postman is a free extension for the Google Chrome browser that helps developers test APIs.

Using your access token in Postman would look something like this:



Permission Scopes

Every access token that is issued has permissions to make certain types of requests to the WPS API – these are called scopes. If you receive an error specifying that you do not have the appropriate scope, this means you are not allowed to access that resource. If you believe this is a mistake and you should have access, please contact us.

Revocation

API access tokens align with account permissions. If your account is deleted or if you are no longer allowed access to our API, your tokens will be revoked.

> Note
>
> API access tokens can only be removed by WPS. As a dealer, or someone working for a dealer, you cannot remove/revoke your own access token. Contact us if you need your API access modified in any way.

Errors

The WPS API uses HTTP status codes.

Our status codes normally fall within three ranges:

- 2xx status codes indicate success
- 4xx status codes indicate a client side error
- 5xx status codes indicate a server side error

For 422 errors, we return validation errors so you can correct the request.

Services

Attributes

Attributes are a quality or feature regarded as a characteristic or inherent part of a parent object.

We apply attributes to Items (and sometimes Products) periodically as they make their way through our system. Attributes are separated into two parts: Attributekeys and Attributevalues. Think of them as a key/value pair that are interdependent and never really used separately.

`Attributekey` : `Attributevalue`

An example of probably the most common Attributekey would be `Color` and an example of a common Attributevalue would be `Red`. Naturally, together that combination would apply to all the items of which we classify as being red in color. Another common attribute example would be "Size : Large" where `Size` is the Attributekey and `Large` is the Attributevalue.

More examples (there are hundreds):

Attributekey	Attributevalue
`Color`	`Red` `Green` `Blue` ...
`Size`	`Small` `Medium` `Large` `Extra Large` ...

Finish	Machined Natural Polished Stainless ...
Material	Aluminum Carbon Kevlar Nickel-Chrome-Molybdenum Plastic ...

Example requests

Attributekeys

Collection

`https://api.wps-inc.com/attributekeys`

Entity

`https://api.wps-inc.com/attributekeys/15`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/attributekeys/1,13,15`

Attributevalues

Collection

`https://api.wps-inc.com/attributevalues`

Entity

`https://api.wps-inc.com/attributevalues/901`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/attributevalues/848,859,870,881,901`

Parent Attributekey

`https://api.wps-inc.com/attributevalues/901/attributekey`

Associated Items

`https://api.wps-inc.com/attributevalues/901/items`

Associated Products

`https://api.wps-inc.com/attributevalues/1049/products`

Associated Vehicles

`https://api.wps-inc.com/attributevalues/679/vehicles`

Blocks

Blocks are essentially a chunk of HTML that we attach to Products.

We use Blocks to display blobs of HTML markup on our websites for things like detailed breakdowns, tables, showcasing certain characteristics, and highlighting aspects of a Product and such. Blocks can have Images associated with them, and the references to those Images are typically also embedded within the HTML data itself.

The `data` property is a string of markup that is commonly included when displaying product detail on the web.

Example requests

Collection

`https://api.wps-inc.com/blocks`

Entity

`https://api.wps-inc.com/blocks/1`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/blocks/1,2,3,4,5`

Associated Images

`https://api.wps-inc.com/blocks/27/images`

Associated Products

`https://api.wps-inc.com/blocks/27/products`

Brands

A Brand is a unique design, sign, symbol, words, or a combination of these, employed in creating an image that identifies a product and differentiates it from its competitors.

In the WPS API, Brands refer to products manufactured by a particular company under a particular name. This endpoint allows you to retrieve a collection of all our Brands, or the data surrounding a specific Brand via the entity endpoint.

Example requests

Collection

`https://api.wps-inc.com/brands`

Entity

`https://api.wps-inc.com/brands/135`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/brands/66,72,135,522`

Associated Images

Get all the Images associated to the "FLY Racing" Brand.

`https://api.wps-inc.com/brands/135/images`

Associated Items

Get all the Items associated to the "FLY Racing" Brand.

`https://api.wps-inc.com/brands/135/items`

Countries

Countries in the WPS API refer to the **Country of Origin** of an Item in an ISO 3166-1 alpha-2 code format.

This endpoint allows you to retrieve a collection of all our Countries, or the data surrounding a specific Country via the entity endpoint. The Countries endpoint also allows you to retrieve the associated relationships that Countries have with other services.

Example requests

Collection

`https://api.wps-inc.com/countries`

Entity

This request will return the entity with details like the Country `name` and Country `code`. Like in this example, the Country **id** of `101` = "Italy".

`https://api.wps-inc.com/countries/101`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/countries/44,101,208`

Associated Items

Get all the Items associated with the Country of "Italy".

`https://api.wps-inc.com/countries/101/items`

Features

Features are a distinctive characteristics, traits, or aspects of a Product.

In the industry, these are typically referred to as the "bullet points" of a product. In previous versions of the API, the description and bullet points were lumped together into one mix-mash of raw strings combined with formatting and other

HTML-style markup that produced *bullet points*. This is not ideal if you try to utilize these on, for example, a website product detail page and style and display them in a meaningful fashion.

In Version 4 of the WPS API, an effort was made to separate the *description* of a product from the *bullet points*. We've cleaned up all of that and now clearly separate a Products description from it's Features.

Example requests

Collection

`https://api.wps-inc.com/features`

Entity

`https://api.wps-inc.com/features/673`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/features/8,1235,442,1820`

Parent Product

Get the Product that a particular feature belongs to.

`https://api.wps-inc.com/features/673/products`

Images

The Images endpoint allows you to retrieve information about images and their relationships to other entities.

Our images are now cataloged and tracked in our database and tied to other entities in a more relational fashion. In previous versions of our API, we simply stored a one-field reference to the filename. But now we store tons of additional data on images in their own database. The information that can be retrieved are things like the alt tags, image MIME type, width, height, size, and static references to the images CDN location on the web.

Example requests

Collection

`https://api.wps-inc.com/images`

Entity

`https://api.wps-inc.com/images/96`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/images/96,52207,49944,18220,51112,48572`

Associated Blocks

`https://api.wps-inc.com/images/49944/blocks`

Associated Brands

`https://api.wps-inc.com/images/96/brands`

Associated Items

`https://api.wps-inc.com/images/18220/items`

Associated Posts

`https://api.wps-inc.com/images/51112/posts`

Associated Products

`https://api.wps-inc.com/images/48572/products`

Associated Tags

`https://api.wps-inc.com/images/1/tags`

Associated Taxonomyterms

`https://api.wps-inc.com/images/49957/taxonomyterms`

Forming a full image URL

I've located the image I want to use, now what?

To create a full URI reference to a particular image on our CDN, you may combine the **domain**, **path**, and **filename** response elements. The last important piece of the full URI reference is the image **style**.

Image Styles

Starting in V4 of the API, every image we upload into our system gets created in four different derivatives:

- `200_max`
- `500_max`
- `1000_max`
- `full`

Those derivatives are simply varying dimensions of the same image.

> **Warning**
>
> The various *styles* are **not** a guarantee of size/dimension. For example, not all images in our system are 1000 pixels by 1000 pixels. So requesting an images `1000_max` image style will not necessarily return a 1000px by 1000px image. The style is simply indicating it's **maximum** dimension (either height or width) of 1000 pixels.

For example, if the API response looks like this:


```
{
    "id": 79468,
    "domain": "cdn.wpsstatic.com/",
    "path": "images/",
    "filename": "6dde-59cd72ea6f409.jpg",
    "alt": null,
    "mime": "image/jpeg",
    "width": 2706,
    "height": 2575,
    "size": 2708162,
    "signature":
"8d3d02758b7b091f5a6acf7b7706ebdf92b92fc3f1c86380b7ada642f6812de9",
    "created_at": "2017-09-28 22:08:53",
    "updated_at": "2017-09-28 22:08:53"
}
```

You would combine the four parts (domain, path, {style}, filename) to create a URI like so...
`http://cdn.wpsstatic.com/images/500_max/6dde-59cd72ea6f409.jpg`

Since in the image URI we specified the `500_max` style, the size of image we received is a proportionally resized 500px by 476px image with it's original ratio preserved.

Lastly, if you do not want to restrict the maximum size of the image at all and you want the fullsize image (the original size we uploaded the image at) you would simply use the **full** image style. Fair warning though, the full image style has absolutely no maximum sizing restrictions so images *could* potentially be up to 5000px or more!

Inventory

This endpoint allows you to retrieve inventory levels.

Inventory levels 25

It is important to note that if we have 25 or more of a particular item, we will just show 25. Inventory levels have a ceiling of 25 so as to avoid disclosing our actual inventory levels beyond this threshold.

Example requests

Collection

`https://api.wps-inc.com/inventory`

Entity

`https://api.wps-inc.com/inventory/3`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/inventory/1,2,3,7,8,10,11`

Available Includes

Associated Item

`https://api.wps-inc.com/inventory?include=item`

Available Filters

Filter by Item

`https://api.wps-inc.com/inventory?filter[item_id]=387`

items

Items are the building blocks of a Product; they are simplest form of a Product and refer to one individual variant (specific configuration) or **SKU** of a Product.

For example:

The *FLY Patrol Jersey* is a Product, whereas the *SKU:370-6402X 2X-Large Black/Grey FLY Patrol Jersey* is an Item.

Products	Items

FLY Patrol Jersey **370-640S** → Small → Black/Grey → FLY Patrol Jersey

370-640M → Medium → Black/Grey → FLY Patrol Jersey

370-640L → Large → Black/Grey → FLY Patrol Jersey

370-640X → X-Large → Black/Grey → FLY Patrol Jersey

370-6402X → 2X-Large → Black/Grey → FLY Patrol Jersey

...

FLY Trekker Helmet	**73-1010XS** → X-Small → Gloss Black → Trekker Helmet **73-1010S** → Small → Gloss Black → Trekker Helmet **73-1010M** → Medium → Gloss Black → Trekker Helmet ... **73-1014XS** → X-Small → Hi-Vis Yellow → Trekker Helmet **73-1014S** → Small → Hi-Vis Yellow → Trekker Helmet **73-1014M** → Medium → Hi-Vis Yellow → Trekker Helmet ...

Example requests

Collection

`https://api.wps-inc.com/items`

Entity

`https://api.wps-inc.com/items/`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/items/216584,22864,72757,220510`

Associated Attributevalues

`https://api.wps-inc.com/items/216584/attributevalues`

Associated Unit-of-measurements

`https://api.wps-inc.com/items/216584/unit-of-measurements`

Parent Country

`https://api.wps-inc.com/items/216584/country`

Associated Images

`https://api.wps-inc.com/items/216584/images`

Parent Product

`https://api.wps-inc.com/items/216584/product`

Associated Tags

`https://api.wps-inc.com/items/216584/tags`

Associated Taxonomyterms

`https://api.wps-inc.com/items/216584/taxonomyterms`

Associated Vehicles

`https://api.wps-inc.com/items/216584/vehicles`

Status Codes

DIR

DIRECT SHIP FROM VENDOR

DSC

DISCONTINUED ITEM

CLO

CLOSEOUT ITEM

NA

NOT AVAILABLE AT THIS TIME

NEW

NEW ITEM

NLA

NO LONGER AVAILABLE

PRE

PRE -RELEASE ITEM (CONTACT REP TO ORDER)

SPEC

SPECIAL ORDER

STK

STANDARD STOCKING ITEM

Item Inventory

Pro Tip: Get inventory levels for an Item in one request

As you've probably discovered, there are many ways of getting inventory for a particular Item. But, to reduce the number of requests needed, it may be helpful to use includes.

`https://api.wps-inc.com/items?filter[sku]=72-7044L&include=inventory`

...or even better would be target the exact Item by using a crutch...

`https://api.wps-inc.com/items/crutch/72-7044L?include=inventory`

This will give you an specific entity response instead of a collection with one result.

Products

A Product is a logical grouping of Items and is at the core of a concept that is new to the V4 API called **product variants**, or **variations of a product**.

A product variant is a specific Item that is grouped with related variants that together form a Product. Variants usually vary from each other in one or more properties (or Attributes). For example, a medium-sized, green shirt with a SKU

of `12345` is one product variant of the shirt product; together size, color, and SKU form one variant.

> Note
>
> It is important to understand the difference between Products and Items. Products are not a tangible, physical object meaning they are not the actual merchandise that a customer would purchase as Products themselves do not have a SKU or Price. Products are merely a mechanism for grouping and ultimately arriving at an Item.

By itself, a Product isn't much more than a *name* and *description*, but as you start gathering up all its associated services you'll soon realize that a Product is at the core of what the WPS API provides.

Example requests

Collection

`https://api.wps-inc.com/products`

Entity

`https://api.wps-inc.com/products/207997`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/products/26,212506,212415,211896`

Associated Attributekeys

`https://api.wps-inc.com/products/207997/attributekeys`

Associated Attributevalues

`https://api.wps-inc.com/products/207997/attributevalues`

Associated Blocks

`https://api.wps-inc.com/products/207997/blocks`

Associated Features

`https://api.wps-inc.com/products/207997/features`

Associated Images

`https://api.wps-inc.com/products/207997/images`

Associated Items

`https://api.wps-inc.com/products/207997/items`

Associated Resources

`https://api.wps-inc.com/products/26/resources`

Associated Tags

`https://api.wps-inc.com/products/207997/tags`

Associated Taxonomyterms

`https://api.wps-inc.com/products/207997/taxonomyterms`

Resources

Resources are outside, external assets that relate to Products and are typically embeddable references to something on the web in some shape or form.

Example Resource types*: Flickr, Imgur, Instagram, Slideshare, Tumblr, Vevo, Vimeo, Vine, YouTube. They could also be references to a PDF hosted on the Internet somewhere, or a URL to a Google Doc.

Room to grow

* Resources are not necessarily limited to any of these *types*. More Resource types could be added as time goes on.

Resources are great for accompanying product detail page information on your website. It is our hope that you can utilize the resources we've gathered up and associated to Products to help sell the them via various sales channels. Resources are very versatile and can be used in a number of ways.

Usage examples

Display a linked thumbnail of the Resource...



Embed the resources as an iframe...

Resources contain a property called `reference` which is typically just the *identifier* of the asset. More often than not it will simply be the YouTube identifier, such as `CPeF9546uac` because there are so many ways to utilize it...

```
https://www.youtube.com/watch?v=CPeF9546uac
https://youtu.be/CPeF9546uac

<iframe src="https://www.youtube.com/embed/CPeF9546uac"></iframe>

http://img.youtube.com/vi/CPeF9546uac/default.jpg
http://img.youtube.com/vi/CPeF9546uac/sddefault.jpg
http://img.youtube.com/vi/CPeF9546uac/mqdefault.jpg
http://img.youtube.com/vi/CPeF9546uac/hqdefault.jpg
http://img.youtube.com/vi/CPeF9546uac/maxresdefault.jpg
http://img.youtube.com/vi/CPeF9546uac/0.jpg
http://img.youtube.com/vi/CPeF9546uac/1.jpg
http://img.youtube.com/vi/CPeF9546uac/2.jpg
http://img.youtube.com/vi/CPeF9546uac/3.jpg
```

> But wait, there's more
>
> There are even more YouTube hostnames that can be used: i.ytimg.com, i1.ytimg.com, i2.ytimg.com, i3.ytimg.com, i4.ytimg.com, or s.ytimg.com.

The other *types* of Resources would also work the same way. We provide the Vimeo or Instagram identifier, it is up to you how you utilize it.

Example requests

Collection

`https://api.wps-inc.com/resources`

Entity

`https://api.wps-inc.com/resources/1`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/resources/268,185,61`

Associated Products

`https://api.wps-inc.com/resources/1/products`

Tags

Tags are are a simple concept that most people are familiar with. Tags are emphasized, superlative words or terms that are associated to Products.

Tags are things like: `Hot Seller`, `Rugged`, `Made in USA`, `Cold Weather`, `Popular`, `Heavy-duty`, `Featured`, `Christmas`, `Universal`, and so on.

Tags allow us to mark certain Products for the purpose of highlighting them, presenting them in a unique fashion, or displaying them in different areas of our sites.

Future enhancement

Currently, we've only applied a few Tags to a few Products. As time goes on we will be associating many more. In the future, we may even consider associating Tags to other services like Blocks, Items, Images, Vehicles, etc.

Example requests

Collection

`https://api.wps-inc.com/tags`

Entity

`https://api.wps-inc.com/tags/25`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/tags/10,7`

Associated Images

`https://api.wps-inc.com/tags/25/images`

Associated Items

`https://api.wps-inc.com/tags/25/items`

Associated Products

`https://api.wps-inc.com/tags/23/products`

Taxonomyterms

Taxonomyterms are a scheme of classification as it relates to the different services in our API. It is an order or arrangement of different Images, Items, Products, or Vehiclemodels into categories.

Taxonomyterms are how we categorize entities into nested structures. In the past, the WPS print catalogs were our primary method of categorization. This is no longer the case with the addition of Vocabularies and their associated Taxonomyterms in Version 4 of the WPS API.

Taxonomyterms are children of Vocabularies and have a one-to-many Vocabulary→Taxonomyterms relationship.

Info

Taxonomyterms utilize a Nested set model that allows us to manage and access hierarchical tree data in a performant manner.

Example requests

Collection

`https://api.wps-inc.com/taxonomyterms`

Entity

`https://api.wps-inc.com/taxonomyterms/197`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/taxonomyterms/1,2,3,4,5`

Associated Images

`https://api.wps-inc.com/taxonomyterms/10/images`

Associated Items

`https://api.wps-inc.com/taxonomyterms/197/items`

Associated Products

`https://api.wps-inc.com/taxonomyterms/76/products`

Associated Vehiclemodels

`https://api.wps-inc.com/taxonomyterms/50/vehiclemodels`

Parent Vocabulary

`https://api.wps-inc.com/taxonomyterms/197/vocabulary`

Pro Tip: Catalog Classification

A very popular Vocabulary is one we call "Catalog Classification" which is used as a very top-level and overall classification to indicate which print catalog an Item belongs to, is currently in, or has ever appeared in. The vocabulary id for this is **15** for which the Taxonomyterms can be seen at `/vocabularies/15/taxonomyterms`.

—

In the context of Taxonomyterms, this means you could retrieve all the Items for a specific Taxonomyterm (For example, id **193** is the **ATV** catalog classification.) `/taxonomyterms/193/items` This is a pretty easy way to get all Items that we classify as "ATV", which is a question we get asked a lot.

Vocabularies

Vocabularies are what determines a set of Taxonomyterms. They are the parent of several Taxonomyterms and acts as the container that holds the grouping of Taxonomyterms inside.

The Vocabulary endpoint allows you to retrieve a collection of Vocabularies, or all the child categories (Taxonomyterms) of a Vocabulary.

Example requests

Collection

`https://api.wps-inc.com/vocabularies`

Entity

`https://api.wps-inc.com/vocabularies/15`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/vocabularies/1,2,3,7,8,10,11`

Associated Taxonomyterms

`https://api.wps-inc.com/vocabularies/3/taxonomyterms`

Pro Tip: Catalog Classification

A very popular Vocabulary is one we call "Catalog Classification" which is used as a very top-level and overall classification to indicate which print catalog an Item belongs to, is currently in, or has ever appeared in. The vocabulary id for this is **15** for which the Taxonomyterms can be seen at `/vocabularies/15/taxonomyterms`.

Warehouses

This endpoint allows you to retrieve a list of all our warehouses.

Example requests

Collection

`https://api.wps-inc.com/warehouses`

Entity

`https://api.wps-inc.com/warehouses/3`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/warehouses/1,2,3,7,8,10,11`

Unit of Measurement

Units of measurement represent how an item is packaged.

Collection

`https://api.wps-inc.com/unit-of-measurements`

Entity

`https://api.wps-inc.com/unit-of-measurements/1`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/unit-of-measurements/1,2`

Restricted Services

Vehicles

The Vehicle endpoint allows you to retrieve a *collection* of all our Vehicles, or the data surrounding a specific Vehicle via the *entity* endpoint.

The association of Vehiclemodels to Vehicleyears form first-class citizens we call **Vehicles**. A Vehicle is just that; the actual, physical machine. It is a veritable dirt bike, or ATV, or snowmobile, or whatever. But even still, it is not just a model that was produced at some point in time, like for example a `CRF250R`; it is a specific year `2014` `CRF250R` dirt bike.

If any exist in our system, you can also determine the Attributevalues that are associated to the Vehicle, such as OEM specifications like: `Bolt Pattern`, `Fuel Capacity`, etc.

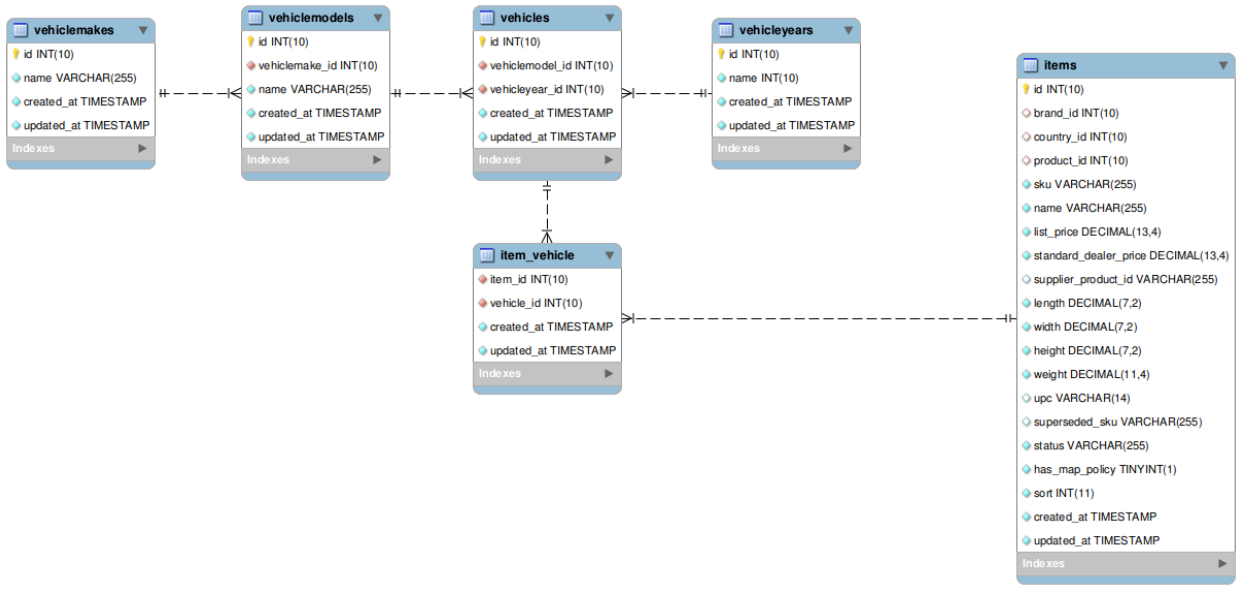
Lastly, if you wish to determine all the Items associated with a particular vehicle, the Vehicles endpoint is where you would make that request.

Fitment access required

Fitment data requires an additional level of authorization. After signing up and being set up in our system, users are able to consume all of our services with the exception of fitment data. In order to make API requests to our fitment services users must contact their sales representative for more information on obtaining access.

Schema

In an effort to clarify the relationship between the various vehicle entities, we've shared this enhanced entity-relationship (EER) diagram illustration.



Example requests

Collection

`https://api.wps-inc.com/vehicles`

Entity

`https://api.wps-inc.com/vehicles/1`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/vehicles/1,2,3,4,5`

Associated Attributevalues

`https://api.wps-inc.com/vehicles/1/attributevalues`

Associated Items

`https://api.wps-inc.com/vehicles/1/items`

Pro Tip

Using the "includes" helper comes in very handy for retrieving the building a database of vehicle information.

Give this request a try:

`https://api.wps-inc.com/vehicles?include=vehiclemodel.vehiclemake,vehicleyear`

With this request, what you are essentially saying is:

"Give me a collection of all Vehicles and include the associated Vehiclemodel and Vehicleyear. With that Vehiclemodel, also join in its Vehiclemake."

Vehiclemakes

A vehicles "make" is generally the company that manufactured the vehicle. Therefore, you guessed it, the Vehiclemakes endpoint provides information on vehicle makes.

The Vehiclemake endpoint allows you to retrieve a collection of all our Vehiclemakes, or the data surrounding a specific Vehiclemake via the entity endpoint. This endpoint can be useful to retrieve the list of vehicle makes such as `Arctic Cat`, `Honda`, `Kawasaki`, `Polaris`, `Suzuki`, etc. which is great for constructing a drop-down menu of "makes" for use on your website.

You could also take this a step further and use includes on this endpoint to gather up the Vehiclemodels associated to each Vehiclemake.

Fitment access required

Fitment data requires an additional level of authorization. After signing up and being set up in our system, users are able to consume all of our services with the exception of fitment data. In order to make API requests to our fitment services

users must contact their sales representative for more information on obtaining access.

Example requests

Collection

`https://api.wps-inc.com/vehiclemakes`

Entity

`https://api.wps-inc.com/vehiclemakes/23`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/vehiclemakes/23,30,31,42,48`

Associated Vehiclemodels

`https://api.wps-inc.com/vehiclemakes/23/vehiclemodels`

Vehiclemodels

Vehiclemodels (or **vehicle model** or **model of vehicle**, and typically abbreviated to just "model") is a particular brand of vehicle sold under a marque by a manufacturer, usually within a range of models, usually of different sizes or capabilities.

A few common examples are: `Bonneville 865`, `CRF250R`, `FXD Dyna Super Glide`, `GSX-R600`, `RZR 800`, `ZR 6000 LXR`, and so on.

The Vehiclemodel endpoint allows you to retrieve a *collection* of all our Vehiclemodels, or the data surrounding a specific Vehiclemodel via the *entity* endpoint.

You can also determine the Taxonomyterms we use to categorize the Vehiclemodels into segments such as `ATV`, `Offroad`, `Snow`, `Street`, etc.

Lastly, if you wish to determine all the Vehicleyears associated with a particular vehicle model, the Vehiclemodels endpoint is where you would make that request.

This service can be useful for building a drop-down menu of "models".

Fitment access required

Fitment data requires an additional level of authorization. After signing up and being set up in our system, users are able to consume all of our services with the exception of fitment data. In order to make API requests to our fitment services users must contact their sales representative for more information on obtaining access.

Example requests

Collection

`https://api.wps-inc.com/vehiclemodels`

Entity

`https://api.wps-inc.com/vehiclemodels/2908`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/vehiclemodels/2908,1280,4275`

Associated Taxonomyterms

`https://api.wps-inc.com/vehiclemodels/2908/taxonomyterms`

Associated Vehicleyears

`https://api.wps-inc.com/vehiclemodels/2908/vehicleyears`

Vehicleyears

Vehicleyears (or **vehicle year** or **year of vehicle**, and typically abbreviated to just "year") is used to describe approximately the year in which a vehicle was produced. (ie. `1995`, `2006`, `2013`, etc.)

The Vehicleyear endpoint allows you to retrieve a *collection* of all our Vehicleyears, or the data surrounding a specific Vehicleyear via the *entity* endpoint.

You can also determine all the Vehiclemodels that are associated with a given Vehicleyear.

This service can be useful for building a drop-down menu of "years".

Fitment access required

Fitment data requires an additional level of authorization. After signing up and being set up in our system, users are able to consume all of our services with the exception of fitment data. In order to make API requests to our fitment services users must contact their sales representative for more information on obtaining access.

Example requests

Collection

`https://api.wps-inc.com/vehicleyears`

Entity

`https://api.wps-inc.com/vehicleyears/61`

...or send multiple ids separated by a comma(,)

`https://api.wps-inc.com/vehicleyears/1,2,3,4,61,62,63,64`

Associated Vehiclemodels

`https://api.wps-inc.com/vehicleyears/61/vehiclemodels`

Helpers

Counts

All the ways to get total record counts of various data.

There are a few ways to get counts. The two primary methods are:

1. ?countOnly=true
2. ?include={entity}:count

These *count* helpers can be used on any endpoint and in combination with filters and other helpers too.

Example requests

countOnly=true

Let's start with a simple one: Warehouses. Get a total count of all Warehouses.

`https://api.wps-inc.com/warehouses?countOnly=true`

Easy, right? WPS has six warehouses so this request returns 6 in the structured JSON response.

Next, why not try a more useful endpoint like getting a total count of all Items?

`https://api.wps-inc.com/items?countOnly=true`

But what about relations?

Let's try getting all the Items for a Product.

`https://api.wps-inc.com/products/212415/items?countOnly=true`

Perhaps you want to combine this functionality with other helper parameters (like Filters) to produce even more powerful queries. Piece of cake!

Get a count of all *FLY Racing* branded Items.

`https://api.wps-inc.com/items?filter[brand_id]=135&countOnly=true`

include={entity}:counts

Let's say we want quickly see how many Items are attached to Product.

Get a total count of all Items in the collection of Product.

`https://api.wps-inc.com/products?include=items:count`

Get a total count of Items for a specific Product.

`https://api.wps-inc.com/products/212415?include=items:count`

Ready for a more complicated one? This is how you would count includes that are filtered. Read more about Filtering Includes. This example request will include the Items count, but filtered by only those Items that have a **list_price** less than $112.96 for a specific Product.

`https://api.wps-inc.com/products/211890?include=items:filter(list_price|112.96|lt)count`

Crutches

In order to make the transition from Version 3 to Version 4 of the WPS API as painless as possible, we've created a clever concept which we've humorously* dubbed "crutches".

** May not contain actual humor.*

Crutches are the bridge from V3 (SKU-based) to V4 (auto-incrementing) identifiers. This feature allows you to "crutch" to the resource using the old V3 identifier, without needing to know the V4 `id` of the resource.

Obviously this only applies to concepts that existed in V3 and are carried forward into V4. Things like Brands and Items.

Examples

Items

In the case of the Highway 21 Small Black *Gunner* Jacket, we can make a request for that specific Item using the V4 `id` with this request:

`https://api.wps-inc.com/items/72728`

Conversely, we can "crutch" to the same Item using the V3 identifier with this request:

`https://api.wps-inc.com/items/crutch/489-1014S`

Attributekeys

In the case of the `Color` Attributekey, we can make a request for that specific Attributekey using the V4 `id` with this request:

`https://api.wps-inc.com/attributekeys/15`

Conversely, we can "crutch" to the same Attributekey using the V3 identifier with this request:

`https://api.wps-inc.com/attributekeys/crutch/3`

Fields

Transform the API response by specifying exactly which fields to return in the response.

The *fields* helper parameter allows you to indicate the properties you would like to be returned in the API response. Doing this not only reduces the overall size of the data transmitted, but it also makes responses more human-readable to omit fields that you don't need or intend on using.

Example requests

Single field

Get all the Items for a specific Product, but only return the `sku` property in the response.

```
https://api.wps-inc.com/products/65184/items?fields[items]=sku
```

Multiple fields (separated by a comma)

Get all the Images for a specific Item, but only return the `filename` and `mime` properties in the response.

```
https://api.wps-inc.com/items/crutch/73-8603L/images?fields[images]=filename,mime
```

Example using the includes helper

Get a collection of Products, with their Items included, but only return the `name` property of the Product and only the `sku` and `list_price` properties of the Items.

```
https://api.wps-inc.com/products?include=items&fields[products]=name&fields[items]=sku,list_price
```

Filters

Filters give the ability to narrow down the results of a request to specific subset of data based on properties of the service.

The parameter syntax structure is: `?filter[field][operator]=value`.

Operators

`[eq]`	Equal*
`[ne]`	Not equal
`[lt]`	Less than
`[lte]` or `[le]`	Less than or equal to
`[gt]`	Greater than
`[gte]` or `[ge]`	Greater than or equal to
`[pre]`	Prefix / Starts with

* Default behavior if an operator is not sent.

The thing to remember about filters is that you can filter by *any* of the properties of a particular service. If you're making a request to the `/items` service, you can filter by things like sku, list_price, name, etc. Any of the properties of the Items entity are available to be used as filters. By the same token, if you are making a request to the `/images` service, you can filter by the properties of THAT service. Things like filename, size, mime, alt, etc. That is why we don't document every single filter that is available. It is/should be a given and assumed. Any property on a particular endpoint is filter-able.

The `[field]` and `value` will depend on the endpoint and what you are trying to filter. The easiest way to explain Filters is to just give examples.

Examples

Bear in mind that these are only a few examples; we can't possibly document examples of every possible filter technique. Be adventurous and experiment on your own to see what you can come up with to meet the needs of your business.

Items

Get all FLY Racing Items.

In this example, only Items with a `[brand_id]` of `135` (FLY Racing) would be returned. As you can see, the operator has not been specified, so `[eq]` is assumed.

```
https://api.wps-inc.com/items?filter[brand_id]=135
```

Get all Items that have a MSRP less than $200.

In this next example, only Items with a `[list_price]` of `[lt]` (less than) `200` dollars would be returned.

```
https://api.wps-inc.com/items?filter[list_price][lt]=200
```

Get all Items that have a MSRP between $200 and $400.

You can also combine multiple filters to produce even more powerful queries.

In this example, only Items with a `[list_price]` of `[gt]` (greater than) `200` dollars and `[lt]` (less than) `400` dollars would be returned.

```
https://api.wps-inc.com/items?filter[list_price][gt]=200&filter[list_price][lt]=400
```

Get all Items where the WPS Item number (SKU) starts with "87-4".

In this example, only Items with a `[sku]` that are `[pre]` (prefixed with) `87-4` would be returned.

```
https://api.wps-inc.com/items?filter[sku][pre]=87-4
```

Get all Items that were updated in the last 2 weeks.

In this example, only Items with a `[updated_at]` date that is `[gt]` (greater than) `2025-05-10` would be returned.

```
https://api.wps-inc.com/items?filter[updated_at][gt]=2025-05-10
```

Products

Get all Products where the name starts with "GM5".

In this example, only Products with a `[name]` that are `[pre]` (prefixed with) `GM5` would be returned.

```
https://api.wps-inc.com/products?filter[name][pre]=GM5
```

Get all the Tags of a particular Product that start with "Popu".

You can also filter the associations of a Product.

In this example, only a Products associated Tags with a `[name]` that are `[pre]` (prefixed with) `Popu` would be returned. Since the following request ultimately returns Tags, what we are filtering is the Tags.

```
https://api.wps-inc.com/products/208016/tags?filter[name][pre]=Popu
```

Pro Tip→ You can use filters on any endpoint

Keep in mind that you can use filters on more than just Items and Products. Filters work on any service throughout the API. You can filter any entity or the associations of that entity with ease.

Filtering Includes

Much like standard filters, filtering includes gives you the ability to narrow down the results of a request to specific subset of data. The difference is what you are operating on. The includes themselves are the part of the response that is being filtered, not the primary service.

For example, when you make a request like `/products?include=items:filter(brand_id|135)`, what you are actually saying is: *Give me all Products, and include their Items, but only return those included Items that are of Brand 135 (FLY Racing)*. You are still going to get **all** Products, that part doesn't change, the only part of the response that changes is the included Items will be filtered down to FLY Racing Items.

The parameter syntax structure is: `?include=service:filter(field|value|operator)`.

Operators

`eq`	Equal*
`ne`	Not equal
`lt`	Less than
`lte` or `le`	Less than or equal to
`gt`	Greater than
`gte` or `ge`	Greater than or equal to
`pre`	Prefix / Starts with

* Default behavior if an operator is not sent.

Examples

In this example, all the Products along with their included Items would be returned. However, the included Items would be filtered down to only the Items that have a `list_price` of `lt` (less than) `200` dollars.

```
https://api.wps-inc.com/products?include=items:filter(list_price|200|lt)
```

In this example, all the Products along with their included Items would be returned. However, the included Items would be filtered down to only the Items that have a `sku` that are `pre` (prefixed with) `87-40`.

```
https://api.wps-inc.com/products?include=items:filter(sku|87-40|pre)
```

Pro Tip→ You can filter includes on any endpoint

Keep in mind that you can use filters on more than just Items and Products. Filters work on any service throughout the API. You can filter any entity or the associations of that entity with ease.

Includes

One of the most practical and convenient functions of the API, *includes* are essentially a way to nest one service within another service based on the relationship they have with each another.

This important feature makes your requests much more efficient and greatly reduces the need to make multiple requests to gather all the pieces of data you desire. For those with a database background, think of includes as a "join". With includes you are essentially joining another related database table to the primary entity and nesting it under each item in the response.

Probably the most commonly used example of this functionality would be to request Products with their associated Items included.

```
https://api.wps-inc.com/products?include=items
```

Instead of retrieving an Item, making note of the `product_id` on the Item, and making another request to get the parent Product; we can simply get the Product and *include* all of it's associated Items.

> Works with any valid relationship
>
> Includes are not limited to parent/child relationships; you can also get child/parent relationships. This allows you to incorporate any parent service within a child service provided they have appropriate relationship to one another.

Using the same idea as the previous example but reversing the parent/child relationship, we can retrieve Items with their associated Product included.

```
https://api.wps-inc.com/items?include=product
```

Pay special attention to singularity versus plurality of this request. A Product can have many Items, but an Item only belongs to one Product.

Multiple includes

You can also combine multiple includes to produce even more powerful queries. Just separate each include with a comma (,).

```
https://api.wps-inc.com/products?include=features,images,items,tags
```

Nested includes

You can take includes a step further and retrieve an includes associations as well. Just separate each association by a dot (.). A request like this will include all the Items associated to a Product, but it will also include the Images associated to those Items as well.

```
https://api.wps-inc.com/products?include=items.images
```

It doesn't stop at one level either. You can essentially include an infinite amount of relationship data on an include! You can go as deep as the relationships go. All you have to do is separate each association by a dot (.).

This request will return all the Items associated to a Product, but it will also include the Images associated to those Items and the Tags associated to those Images.

`https://api.wps-inc.com/products?include=items.images.tags`

Keep in mind that this is just an example to help illustrate the usage of Nested includes. In reality, deep relationships like that are few and far between. Currently we don't have many (if any) Tags associated to Images but hopefully this example will help you grasp the concept.

Collection or Entities

Includes work the same on *collection* or *entity* requests. If we were to request one particular Product and include all it's Taxonomyterms, the request would look something like this:

`https://api.wps-inc.com/products/207976?include=taxonomyterms`

More Examples

Get a collection of Items and include the Images associated with each of them.

`https://api.wps-inc.com/items?include=images`

Get a collection of Attributekeys and include the Attributevalues associated with each of them.

`https://api.wps-inc.com/attributekeys?include=attributevalues`

Get a collection of Attributevalues and include the **one parent** Attributekey associated to each of them.

`https://api.wps-inc.com/attributevalues?include=attributekey`

Pagination

The WPS API utilizes a technique called "cursoring" to paginate large result sets.

Cursoring allows deep paging of massive data sets without sacrificing performance. Cursoring separates results into pages (the size of which are defined by the `page[size]` request parameter) and provides a means to move backwards and forwards through these pages.

Paging through the results is accomplished by sending a `page[cursor]` parameter in your requests. Cursors are provided at the bottom of the response in the `meta` property in an object containing defined `current`, `prev`, or `next` properties.

To retrieve cursored results, you initially avoid sending a cursor parameter to the endpoint. By default, the endpoints will assume `page[cursor]=null` was passed as the cursor if you do not provide one.

The `next` value is the cursor that you should send to the endpoint to receive the next batch of responses, and the `prev` is the cursor that you should send to receive the previous batch.

You will know that you have requested the last available page of results when the API endpoint responds with a `next` = **null** in the cursor object. This makes a standard `while` loop in the programming language of your choice an ideal mechanism for making several requests in succession. Just keeping looping *while* the `next` property is not `null`.

Requests

`https://api.wps-inc.com/products`

Note that we did not send a `page[cursor]` initially.

Response (truncated)

```
{
      "data": {
            ...
      },
      "meta": {
            "cursor": {
                  "current": "mZOYdKDe5K49",
```

```
                    "prev": null,
                    "next": "lqjMdY6e6Z07",
                    "count": 10
                }
        }
}
```

We now have a means to move forwards through our data set, via the `next` cursor. Also note, the `prev` property is `null` because we're at the first page.

To retrieve the next page of results, we can send the `next` cursor we just got on our subsequent request like so...

`http://api.wps-inc.com/products?page[cursor]=lqjMdY6e6Z07`

...And the next page of results is returned. Keep advancing until the `next` cursor is `null`, which indicates that there are no more remaining pages.

Pro Tip→ You can use cursors on any endpoint

Keep in mind that you can use cursors on more than just Products. Cursors work on any endpoint throughout the API.

Sorting

The sorting helpers allow you to control the arrangement of the response data in a prescribed sequence.

Arrange the API responses in the order that you wish to receive it by sending a *sort* parameter in your requests.

The parameter syntax structure is: `?sort[direction]=field`.

Pro Tip: [asc] is assumed if direction is left off

The [direction] portion is optional and [asc] is assumed as default if omitted from your request.

For example: `?sort=field` is the same as saying `?sort[asc]=field`

Example requests

Sort by **name** ascending

Get a collection of Attributekeys sorted in alphabetical order by `name`.

```
https://api.wps-inc.com/attributekeys?sort[asc]=name
```

or

```
https://api.wps-inc.com/attributekeys?sort=name
```

Note that both these requests return the same result, sorted in the same way, because [asc] is the default behavior.

Sort by **name** descending

Get a collection of Attributekeys sorted in reverse-alphabetical order by `name`.

```
https://api.wps-inc.com/attributekeys?sort[desc]=name
```

Sort a primary collection based on it's relation field

Get a collection of Products that are sorted based on it's child Items `list_price`.

```
https://api.wps-inc.com/products?include=items&sort[desc]=items.list_price
```

Sorting includes

Include all the Items of a specific Product and sort those Items by `list_price` in descending order highest to lowest.

```
https://api.wps-inc.com/products/187138?include=items:sort(list_price)
```

As with primary collections, if the sort *direction* is left off, we'll assume you want `asc` order.

You can also explicitly pass a *direction* by adding a pipe character (|) between the field and the direction.

```
https://api.wps-inc.com/products/187138?include=items:sort(list_price|desc)
```

Order Processing Overview

The Order Processing web services allow dealers to transmit orders dynamically.

Submit orders and even drop ship them. Just like you, we want to stay in front of the competition. Automation is a key to be competitive in our industry. Order Processing enables you to rapidly fulfill customer orders. A web-based shopping cart or POS system can integrate with this service.

Order Processing has two primary components

- Carts
- Orders

First, send a request to create a cart. After successful, add items to the cart. Once ready, submit that cart as an order. The cart will be processed exactly as if it were submitted through the wpsorders.com ordering system. The order status

can then be requested. Please take look at the links above to explore the options and responses of these services.

Before you begin

PO numbers should be unique for each order placed. Reusing a PO is allowed but it is not ideal. A reused PO will make related orders more difficult to query.

We're always looking for dealer feedback and feature requests. If you have a feature that you need in order to process orders, feel free to ask. We're here to help you. If you are experiencing technical difficulties, please review the documentation. Quite often the answer will be there.

Carts

Create a cart, add items, and then submit. Use the dealer defaults, or specify cart options.

Create a Cart

`POST https://api.wps-inc.com/carts`

Arguments

po_number	A string identifier for the cart. Used to add items and submit the order.

string

	Required
default_warehouse string	The default warehouse. • CA = Fresno, CA • GA = Midway, GA • ID = Boise, ID • IN = Ashley, IN • PA = Elizabethtown, PA • TX = Midlothian, TX
ship_via string	The shipping method code. • AH2D = FedEx 2-Day (AK/HI only) • BEST = Best Ground method available • FE1D = Fedex 1 Day • FE2D = Fedex 2 Day • UP1D = UPS 1 Day Red • UP2D = UPS 2 Day Blue • US1C = USPS Priority Mail • US4C = USPS Parcel Post
cross_ship_via	The cross ship method code. Available options are found under ship_via.

string

ship_to — A 7 character ship to #, assigned by WPS.

string

ship_name — The drop ship recipient.

string (max 30)

ship_address1 — The drop ship address line 1.

string (max 30)

ship_address2 — The drop ship address line 2.

string (max 30)

ship_address3 — The drop ship address line 3.

string (max 30)

ship_city	The drop ship address city. string (max 17)
ship_state	The drop ship address state. string (max 2)
ship_zip	The drop ship address zip code. string (max 15)
ship_phone	The drop ship phone number. string (max 15)
email	The drop ship email address. string (max 50)
comment1	The comment (line 1).

string (max 50)

comment2	The comment (line 2)

string (max 50)

allow_backorder	Allow back order.

boolean

hold_order	Hold order.

boolean

multiple_warehouse	Fill from multiple warehouses.

boolean

proof_of_delivery	The signature is required.

boolean

promo_code	The promo code. string
pay_type	CC = credit card, OO = open order string
cc_last_four	The last four digits of the credit card on file to use. integer
description	The cart description. string (max 50)

Response

cart_number	The unique number assigned to the cart. string

po_number	PO number used when cart was created. string
shipment_type	The shipment type. string

Show Cart

```
GET https://api.wps-inc.com/carts/{po_number}
```

Add Items to Cart

```
POST https://api.wps-inc.com/carts/{po_number}/items
```

Arguments

item_sku or item_id	The Item SKU or Item ID can be used. string Required
quantity	Item quantity desired

integer	Required
note string (max 30)	A note.

Response

sku string	The item SKU.
name string	The item name.
available_quantity integer	The available quantity.
backorder_quantity integer	The backorder quantity.
dealer_price	The actual dealer price.

float

Delete Cart

`DELETE https://api.wps-inc.com/carts/{po_number}`

Orders

Order can be created and then the status of that order can be retrieved.

Create an Order (Submit Cart)

`POST https://api.wps-inc.com/orders`

Arguments

po_number	PO number used when cart was created.		
string	Required		

Response

order_number	The unique number assigned to the order.

string

Order Status

Get a PO with orders.

`GET https://api.wps-inc.com/orders/{po_number}`

Or get multiple POs with orders.

`GET https://api.wps-inc.com/orders`

Arguments

from_date `YYYYMMDD`	Find orders created on or after this date

string

Required

to_date `YYYYMMDD`	Find orders created on or before this date

string

Required

The PO Object

po_number string	PO number used when cart was created.
ship_name string	The shipping name.
ship_address string	The shipping address.
ship_city string	The shipping city.
ship_state string	The shipping state.
ship_zip string	The shipping zip.

order_details array	Each order's details: • order_number: The unique number assigned to the order. (string) • order_status: The status of the order. (string) • invoice_number: The unique number assigned to the invoice. (integer) • freight: The cost of freight. (float) • misc_charges: The miscellaneous changes. (float) • order_total: The total order cost. (float) • warehouse: This warehouse fulfilled the order. (string) • order_date: The date the order was placed. (string) • ship_date: The date the order was shipped. (string) • invoice_date: The date the order was invoiced. (string) • ship_via: The shipping method code. (string) • tracking_numbers: An array of tracking numbers assigned to the order. (array) • items: An array of the items for the order. (array)

Possible ship_via Response Types

- AH2D = FedEx 2-Day (AK/HI only)
- BEST = Best Method
- CO5 = OnTrac Ground
- CPU = Customer Pickup
- FDXG = Fedex Ground
- FDXH = Fedex Home Delivery
- FDXP = Fedex SmartPost > 1lb
- FDXS = Fedex SmartPost <= 1lb
- FE1D = Fedex 1 Day
- FE2D = Fedex 2 Day
- FLTS = Fleet Street
- GRND = Ground Delivery
- GSCP = GSO Ground

- SPDY = Speedy Metro
- TRK = Truck
- UPS = UPS Ground
- UPSP = UPS SurePost
- UPSR = UPS Ground Residential
- UP1D = UPS 1 Day Red
- UP1W = UPS Saturday Delivery
- UP2D = UPS 2 Day Blue
- USFC = USPS 1st Class Mail 1oz to 16oz *ONLY
- US1C = USPS Priority Mail
- US4C = USPS Parcel Post
- 1DAY = 1 Day Air Delivery
- 2DAY = 2 Day Air Delivery
- 3DAY = 3 Day Air Delivery

Possible order_status Response Types

- Invoiced
- In O/E
- Shipped
- Picking
- B/O
- Checking

Response examples (limited to 1 for example sake):

Attributes


```
{
  "data": [
    {
      "id": 1,
      "name": "Product Type",
      "created_at": "2016-06-17 20:53:19",
      "updated_at": "2025-03-11 21:28:50"
    }
  ],
```

```
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Brands


```
{
  "data": [
    {
      "id": 1,
      "name": "509",
      "created_at": "2016-05-04 19:22:46",
      "updated_at": "2016-05-04 19:22:46"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
```

```
}
```


Countries


```
{
  "data": [
    {
      "id": 1,
      "code": "AD",
      "name": "Andorra",
      "created_at": "2016-05-04 19:22:10",
      "updated_at": "2016-05-04 19:22:10"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Features


```
{
  "data": [
    {
      "id": 45,
      "product_id": 50608,
```

```
      "icon_id": null,
      "sort": 5,
      "name": "Comfort collar and wrist cuffs",
      "created_at": "2016-06-22 23:17:45",
      "updated_at": "2016-08-04 18:48:16"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Images


```
{
  "data": [
    {
      "id": 1,
      "domain": "cdn.wpsstatic.com/",
      "path": "images/",
      "filename": "1ee1-572a4c0b962a5.jpg",
      "alt": null,
      "mime": "image/jpeg",
      "width": 120,
      "height": 38,
      "size": 1988,
```

```
      "signature":
"6e31167e6aa503c9a678327e4c174cbd995ee40f9a11e45efc6cf50ab
3e98298",
      "created_at": "2016-05-04 19:22:51",
      "updated_at": "2017-02-23 18:00:23"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Inventory


```
{
  "data": [
    {
      "id": 1800,
      "item_id": 585114,
      "sku": "71-15014",
      "ca_warehouse": 0,
      "ga_warehouse": 0,
      "id_warehouse": 25,
      "in_warehouse": 25,
      "pa_warehouse": 0,
      "pa2_warehouse": 0,
      "tx_warehouse": 25,
      "total": 75,
```

```
      "created_at": "2020-09-09 22:35:20",
      "updated_at": "2025-05-24 01:01:13"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


items


```
{
  "data": [
    {
      "id": 387,
      "brand_id": 406,
      "country_id": null,
      "product_id": 6,
      "sku": "015-01001",
      "name": "MULTIRATE FORK SPRINGS 35MM",
      "list_price": "126.95",
      "standard_dealer_price": "92.18",
      "supplier_product_id": "FS-1017",
      "length": 24.5,
      "width": 4.7,
      "height": 1.9,
      "weight": 2.46,
      "upc": null,
```

```
      "superseded_sku": null,
      "status_id": "NA",
      "status": "NA",
      "unit_of_measurement_id": 12,
      "has_map_policy": false,
      "sort": 0,
      "created_at": "2016-06-17 20:47:51",
      "updated_at": "2025-03-27 12:39:05",
      "published_at": "2016-06-17 20:47:51",
      "product_type": "Suspension",
      "mapp_price": "0.00",
      "carb": null,
      "propd1": null,
      "propd2": null,
      "prop_65_code": null,
      "prop_65_detail": null,
      "drop_ship_fee": "FR",
      "drop_ship_eligible": true
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Products

```
{
  "data": [
    {
      "id": 6,
      "designation_id": null,
      "name": "Multirate Fork Springs Kit",
      "alternate_name": null,
      "care_instructions": null,
      "description": "Our Multirate fork springs are produced from the best spring materials available. They are precision wound and stress relieved and mechanically polished to increase surface density to improve corrosion resistance. Straight rate springs have the same rate every inch that they are compressed, which is fine on flat super smooth roads but they lack the ability of our Multirate fork springs to handle some of the more unpredictable road conditions without sacrificing ride quality. If you have a heavier bike or increased load, try our new “GENISIS SERIES” fork springs.",
      "sort": 0,
      "image_360_id": null,
      "image_360_preview_id": null,
      "size_chart_id": null,
      "created_at": "2016-06-17 20:47:51",
      "updated_at": "2025-03-27 12:44:37"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
```

```
  }
}
```


Resources


```
{
  "data": [
    {
      "id": 1,
      "name": "Bolt Motorcycle Hardware",
      "type": "youtube",
      "reference": "YP1i9hkSYn8",
      "created_at": "2017-01-13 22:52:34",
      "updated_at": "2017-01-13 22:52:34"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Tags


```
{
  "data": [
    {
      "id": 1,
```

```
      "name": "Vests",
      "slug": "vests",
      "created_at": "2016-06-23 15:39:57",
      "updated_at": "2016-06-23 15:39:57"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Taxonomyterms


```
{
  "data": [
    {
      "id": 1,
      "vocabulary_id": 2,
      "parent_id": null,
      "name": "Plow",
      "slug": "plow",
      "description": null,
      "link": null,
      "link_target_blank": false,
      "left": 45,
      "right": 58,
      "depth": 0,
      "created_at": "2016-06-22 21:39:48",
```

```
      "updated_at": "2021-08-27 18:04:25"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```


Vocabularies


```
{
  "data": [
    {
      "id": 1,
      "name": "HDTwin Products",
      "description": "Product taxonomy for the hdtwin.com consumer
site.",
      "created_at": "2016-06-17 21:57:34",
      "updated_at": "2016-07-11 16:10:33"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
```

```
    }
  }
}
```


Warehouses


```
{
  "data": [
    {
      "id": 1,
      "db2_key": "ID",
      "name": "Boise",
      "created_at": "2017-03-01 17:49:23",
      "updated_at": "2017-03-01 17:49:23"
    }
  ],
  "meta": {
    "cursor": {
      "current": "61poYD9eaDkR",
      "prev": null,
      "next": "qbWBe8Xe23DZ",
      "count": 1
    }
  }
}
```