Posted in: API (/spaces/127/sandpiper/forums/5044/api)

The DataObject

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/23/2019

Introduction

The Sandpiper `DataObject` (aka "product-data-object", "sandpiper-object" or just plain "object") has been described in several discussions and presentations, but we really haven't formalized its definition. Here is one such reference:

> "Sandpiper further only concerns itself with *product data*, the information about the fundamental characteristics and uses of a product itself. Product data **defines** the product. It is nearly *stateless*; once added, it is only expected to change infrequently."
> — Sandpiper Framework (AUG 6, 2019)

Payload

Because Sandpiper is designed to handle any type of standardized data, its payload structure is a "black-box" string of characters (probably Base64 encoded).

Metadata (Attributes)

Several attributes are used to describe the data object. Current thinking includes:

Field	Type	Description
id	string	unique id of dataobject
owner_id	string: uuid	unique id of creator
payload_type	enum	["aces", "partspro", ...]
payload	string	encoded

Granularity (payload_type)

Sandpiper will define a list of known payload types including one for each of the existing industry standards. What we haven't discussed, however, is at what granularity these are defined. For example, can the payload for "aces" include an entire ACES xml file? If so, how big a file could this be? Could the payload represent an external reference (e.g. url or s3 bucket)? If so, should we standardize this designation?

I think it might be useful to add more granular payload_types. Maybe something like "aces-mmy-parttype" which would represent all applications for a BaseVehicle/PartType. Or, since this is product data, maybe we should consider forcing all payloads to contain only a single part number? Once we understand the requirements, we can tweak the data model to maybe include other metadata making filtering easier (i.e. payload keys).

Thoughts?

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/24/2019

I am personally against an external reference to a full file. That feels like a mildly automated version of the full-file nonsense we have already. I think if we are going to implicitly condone full-level Content trades, it needs to be completely inside the API framework so we encourage buy-in to an M2M (https://en.wikipedia.org/wiki/Machine_to_machine) mindset. Think of it like letting the teenagers drink at home because they are safer with responsible adults around :).

I also like the idea of locking granularity at the part-number level. Otherwise, the granularity expectations in the ecosystem would creep back towards "full". I could live with opening the granularity up to partterminology and/or basevehilce - because they are well-defined standards

that guard against the *creep-towards-full*. I don't have experience to comment on partspro or techdoc granularity.

So here is my payload type list so far:

- aces-app
- aces-apps-for-item
- aces-apps-for-parttype
- aces-apps-for-basevehilce
- aces-apps-for-basevehilce-parttype
- pies-item
- pies-items-for-parttype (multiple pies items that for a specific parttype)
- asset-jpg-filedata (base64 encoding of binary file contents)
- buyersguide-item (human-readable descriptive list of vehicles applied to an item)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5134&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/24/2019

> "I am personally against an external reference to a full file. That feels like a mildly automated version of the full-file nonsense we have already."

I agree. I only included it because of the reference (https://autocare.communifire.com/spaces/127/sandpiper/forums/general/4998/project-requirements-supporting-an-api?postid=5040#5040) in "Level 1" compliance.

We should consider limiting the number of payload-types (in the first release). Each type represents a condition the (receiving) internal-PIM needs to handle. The sending internal-PIM can just pick what works best for them, but the receiver must accept everything we define.

Also, for the following ACES ones, I wonder if they are too high a level (?)

```
aces-apps-for-parttype
aces-apps-for-basevehicle
aces-apps-for-basevehicle-parttype
```

Maybe they should really be by item (i.e. part number) as well. This would make these payload-types more granular. So, for example:

```
aces-item-for-parttype (one partnumber spanning all make/models/years...)
aces-item-for-basevehicle (one partnumber spanning all parttypes...)
aces-item-for-basevehicle-parttype (one partnumber spanning all regions...)
```

On the PIES side, `pies-item` could get really large, and `buyersguide-item` seems more like an aggregation than an atomic delivery.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5135&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/24/2019

My thought on buyer's guide is that it tends to be the artist's rendition of the actual list of apps - not something that can be rendered by the receiver based on an algorithm. I could take it or leave it.

The *pies-item* payload would be a single "item" segment from a PIES file. I guess you could go more granular and say a single segment of a single item (interchange, attributes, digital-assets, etc). I feel like the item is the natural atomic unit of a PIES file. This does not however address a price sheet or marketing copy conveyed via PIES.

I think you are onto something about payload type proliferation. In the interest of keeping V1 simple to implement, maybe we should limit it to the bare minimum payloads - which in my opinion are app, item and asset.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5136&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/24/2019

> "This does not however address a price sheet or marketing copy conveyed via PIES."

Good point... there's most certainly non-item level content in PIES (and non-app information in ACES) that we need to consider as well.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5137&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/25/2019

Does Sandpiper obsolete the concept of net-changes in the underlying payload data? In other words, does it make sense to deliver a "D" ACES App or a PIES Item with MaintenanceType of "A", "C" or "D"?

I've been thinking (lately) that Sandpiper defines the "current state" of the primary data pool (not the transactional state). This seems to be in conflict with the concept of sending net-changes.

In the case of PartsPro, for example, net-changes are generally expected. So I might need to rethink my "current state" model. But if that's true, how could NAPA ever request a full file? Hmmm.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5138&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/25/2019

My opinion is that it absolutely does obsolete the "change" action in an ACES app. I think it does not make sense to deliver anything but an "A" through Sandpiper. The "D" is conveyed by oid at the layer above the black box. For the reason that I can't reliably know what exact object data exists in the receiver - the current-state approach says: "*If you happen to have FLMKwd2zPbh10m, please delete it.*" The only way I could possibly know how to reliably build a "D" ACES app would be to keep a record of ALL the actual raw apps that had EVER been sent to ALL receivers. This is

why I hold the view that net-changes were a fool's errand from the beginning. Obviously if you cater to a single receiver (like NAPA) that kind of record-keeping is possible but still arduous. The way I see it, the "current state" approach is the only viable approach the moment you break away from full files. I think NAPA needs to weigh-in on what is possible in their world while we are still hashing out these fundamentals.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5139&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/27/2019

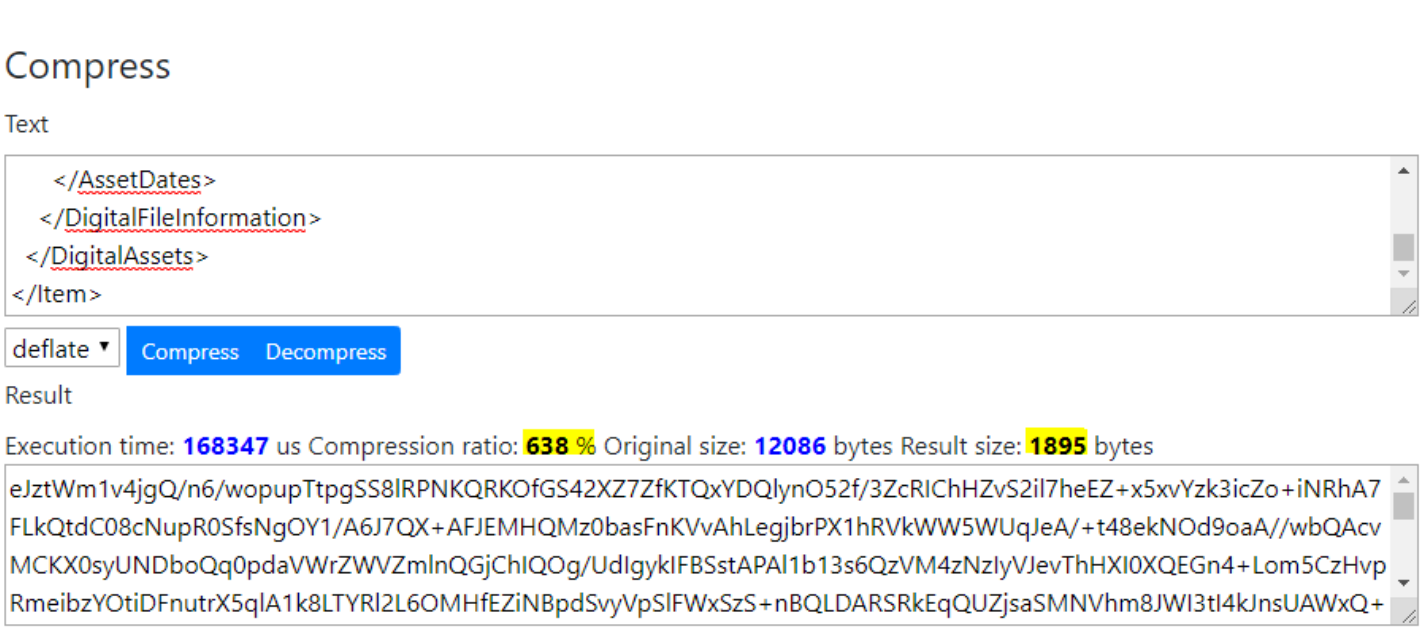
Granularity Discussion (Continued)

With regard to PIES payloads, I think requiring delivery at the **item/segment** level is a good idea because it allows the automation of finer-grained net changes (as opposed to sending the entire **Item**).

With item/segment, for example, you'd send all the interchanges (if one changed) instead of the entire Item just because you had one interchange change. The PIM would need to keep track of changes at a lower level, however.

But it's really a trade-off. A higher level (e.g. Item) is possibly easier for the PIM and produces less objects to track in the data-pool. But those objects are more likely to change, and the payload sizes would be much larger.

In order to get a feel for payload sizes at the Item level, I looked at a few Items from an actual PIES file. The Item segment in that file averaged ~12KB of raw xml. When that data is compressed (with the "deflate" algorithm), the size was about 15% of the original:



When the compressed data was Base64 encoded, it grew to 2524 characters (or about 20% of the original).

These numbers show that sending an entire Item might not be a problem if that's the approach we take, but admittedly the sample size was very small. The examples I looked at had only about 4 interchanges per Item, for example.

If we select item/segment granularity, we also need to find a way to deliver the **item keys** (which in this case is just the "PartNumber") because those are only found at the pies-item level. We should probably make these payload-keys available for any type of delivery.

So, for example, a result might look something like this:

```
{
     "header": {"slice-id":"BPI-WHI", "index": 1, "count": 5000},
     "objects": [
         {
             "oid": "GQBFRvf112p33Q", "type": "PartsProApp", "created": "2019-08-1
2T11:30:00Z",
             "keys": [ "P1234" ],
             "payload": "MQkxOTk1CTEJNQkJCQkJCQkJCQk3NDE4MglNR0MJNDU3M ..."
         },
         {
             "oid": "GQBFRvf112px83a", "type": "PartsProApp", "created": "2019-08-
14T09:30:10Z",
             "keys": [ "P4321" ],
             "payload": "MQkxOTk1CTEJNQkJCQkJCQkJCQk3NDE4MglNR0MJNDU3M ..."
         }
     ]
 }
```


Notice that the result references the slice as well as providing a way to break the response into smaller chunks (the actual implementation might use http range headers instead).

Like

Reply (/forums/post?tid=5128&ReplyPostID=5145&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
9/29/2019

Sandpiper maybe be the opportunity to finally merge the elements of ACES & PIES into one payload. Call it "PACES" for lack of a better name. Many of us in the AutoCare community have unofficially kicked this idea around over the last few years. I don't claim that this is a new idea of mine. Keep in mind that "PACES" brings zero value by simply jamming PIES xml and ACES xml into one container - the value is in a schema that forces us to completely mesh the two in a way that they are no longer un-related feeds of content.

The atomic unit of AutoCare-oriented data in Sandpiper could be a *PACESitem*. The key concept would be that applications are an attribute of the Part - Instead of a stand-alone blob of data divorced from the other product attributes (like PAdb elements, interchange, digital assets, expi, prices, etc.). Envision a traditional PIES *Item* segment with an additional section called "Apps" that would contain all the App records that would have claimed that *Part* in the ACES file paradigm.

From where I stand, this granularity is the Goldilocks (just right) level. A typical brake pads product line is about 2,000 Parts and 100,000 ACES Apps. Sandpiper would break this into 2,000 grains in the pipe. Each grain would contain the entire universe of a SKU including its 50-ish vehicle fitment elements. Every SKU that got new carry-up coverage applied would trigger a re-transmit on the next sync. It would not be as bandwidth efficient as App-level granularity, but it would be easier for the humans to keep their minds around.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5186&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/7/2019

Object Granularity

I understand the appeal of **Luke Smith (https://autocare.communifire.com/people/autopartsource)** 's Goldilocks principle for the definition of what should be contained in a Sandpiper Object payload (above) (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5128/the-dataobject?postid=5186#5186). As I understand his concept, a Sandpiper Object would contain everything related to a finished good (i.e. SKU). This includes all product and application information (combining both PIES and ACES content).

There are several problems I see with this level of granularity, however:

1. **Large payload sizes.** One "Item" in the PIES 6 sample file is 14K (of xml), that's not terrible, but I've seen much bigger Item segments due to huge interchange counts. Looking at (an admittedly large) Piston-Ring ACES file, one popular part (and their were many like it) had 5K apps. Multiply that times 500 xml App characters and you have 2.5MB. Yes, gzip compression should help with this, but it is something to consider.
2. **Object Churn.** The higher the object payload (such as SKU), the more likely the average object lifetime will be short. Unless the part is obsolete, there will be continual maintenance on it. That part will get new pricing, new yearly coverage, and (unfortunately) vcdb-change related updates. This high percentage of change might be the Sandpiper equivalent of "full file" updates.
3. **Channel Differences.** Most suppliers deal with multiple sales "channels", each with their own subtle differences. The AutoZone channel, for example, might require different part numbering and MfrLabels, while the AdvanceAuto channel might require their own set of

user-defined PIES Attributes. If the object payload is at the SKU level, you would probably need a completely different set of objects for each sales channel. (Maybe this is not a bad thing???).

For these reasons, I would suggest a more granular payload. Maybe simply separating PIES from ACES would be "just right". That also accommodates other application delivery standards (e.g. PartsPro) more easily (because a change in product data is separated from a change in each of the application data formats).

goldilocks (/spaces/127/sandpiper/searchresults?keyword=%22goldilocks%22&searchtags=1)
granularity (/spaces/127/sandpiper/searchresults?keyword=%22granularity%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5236&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
10/8/2019

Doug, I completely agree with your points. If we collectively conclude that the industry would tolerate app-level object granularity, let's push for it to be part of the 1.0 spec. We can define other payload types (granularity levels) in future versions. I propose that we start with "ACESapp" and "PIESitem" and specify their schema's as close as possible to ACES and PIES.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5238&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/8/2019

I liked your courser-grained PACES concept (which did **not** include app-level granularity). I just didn't like the idea of combining ACES and PIES data together in a single Sandpiper object.

I'm really liking the idea of PartNumber granularity, though (and maybe having BaseItemID (B05) to PartNumber table for sales channels, but I'm still chewing that one over). So an `aces-item` object-type would have all applications for a single PartNumber, and a `pies-item` object-type would have all PIES segments for a PartNumber.

The non PartNumber stuff (like MarketCopy and PriceSheets), would be included in the data-pool with separate object-types.

I think it's probably more important to make PIM integration easier than to make payload size smaller. I have nothing to back it up, but it seems to me it would be easier for a PIM to track changes at the PartNumber level than at the application or package level. We should look at this more closely. Maybe talk with a couple of the PIM suppliers.

But as you said, we can always add lower levels in the future. That might be necessary for true "real-time" updates.

partnumber (/spaces/127/sandpiper/searchresults?keyword=%22partnumber%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5241&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/31/2019

Answered

Here is my current thinking for Data Object (aka "grain") granularity:

payload-type	key	level	description
aces-file	none	1	full file
pies-file	none	1	full file
partspro-file	none	1	full file
asset-file	asset-id	1	full file (not meta-data)
aces-item	part-number	2	all applications for a part-number
pies-item	part-number	2	all product data for a part-number
pies-marketcopy	company	2	all market-copy for a company
pies-marketcopy-asset	company	2	digital-asset meta-data for marketing (non-item)
pies-pricesheet	pricesheet-num	2	a single pirce-sheet (defined by pricesheet number)
partspro-item	part-number	2	all applications for a part-number

Notice that it's not very granular (lowest level is part-number). Also notice that "asset-file" does not contain meta-data. This means that you'd need a "pies-file" or "pies-item" to define meta-data for an "asset-file".

The "key" would be contained in the data-object as a data column.

data-object (/spaces/127/sandpiper/searchresults?keyword=%22data-object%22&searchtags=1)
granularity (/spaces/127/sandpiper/searchresults?keyword=%22granularity%22&searchtags=1)
keys (/spaces/127/sandpiper/searchresults?keyword=%22keys%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5297&SpaceID=127) Not answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
10/31/2019

Doug, are you opposed to "aces-app" granularity?

Like

Reply (/forums/post?tid=5128&ReplyPostID=5300&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/31/2019

I thought you were supporting Item-level delivery (via the whole PACES Goldilocks thing).

My original thinking was to offer lots of low-level combinations:

```
aces-item-for-parttype (one partnumber spanning all make/models/years...)
aces-item-for-basevehicle (one partnumber spanning all parttypes...)
aces-item-for-basevehicle-parttype (one partnumber spanning all regions...)
```

But that quickly gets out of hand because both sides need to support each type offered.

Now that we're formalizing the Sandpiper "Levels", maybe it makes sense to go fine-grained in Level 3 (real-time).

(I love saying "that's out of scope"... It makes designing a system so much easier!)

out-of-scope (/spaces/127/sandpiper/searchresults?keyword=%22out-of-scope%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5301&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
10/31/2019

"Out-of-scope" is certainly a reasonable answer at this point.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5303&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
10/31/2019

Actually, indulge me briefly on aces-app granularity: In your table above, what would the "key" be for aces-app? In my mind, the answer to that question is the fundamental problem we're here to solve.

Like

Reply (/forums/post?tid=5128&ReplyPostID=5304&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/31/2019

Since an ACES application is the lowest level deliverable, I don't think it would need a key. If I were writing the PIM, I'd probably use a uuid for the App "id", but I'm not sure that would need to be exposed (although it wouldn't hurt anything). The Sandpiper object-id would still be the way the sync is performed.

Is that similar to what you were thinking?

Like

Reply (/forums/post?tid=5128&ReplyPostID=5305&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
10/31/2019

Absolutely. We are on the same page, and it will be in-scope when we get to level 3 :)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5306&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
11/18/2019

In working on the api, I came across the problem of **data ownership** and whether or not we need a "sharing" attribute on the `data-object` ("public" or "private").

The reason this came up is that I'm thinking it could simplify development if both sender and receiver data-objects were contained in the same database (I know, don't say it, this is what you envisioned from the start).

With this mixed approach, we should probably add an **"owner" attribute** and include the ability to keep that data-object from being shared with others. For example, you might not want customer pricing to be shared (i.e synced with a 2nd-degree Sandpiper client without your permission, or by mistake). OE Interchanges are also considered confidential by many suppliers.

And there's the problem. If we keep data-object granularity at the item-level, we can't limit anything lower than an entire pies-item.

I see three solutions:

1. Change the data-object's level of granularity to something smaller than "item".
2. Make all data-objects "private", forcing the receiver to "sanitize" and re-create any data-objects it wants to publish.
3. Have the PIM create "public" and "private" versions of each item that requires it.

I'm torn between #2 and #3. I like the simplicity of "all data-objects are private". It also makes it very clear that you're responsible for the information you pass along (with a separate contract between trading partners).

The third option, though, goes along with my new favorite saying "That's the PIMs responsibility!"

Any thoughts?

ownership (/spaces/127/sandpiper/searchresults?keyword=%22ownership%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5341&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
11/30/2019

Answered

Once again, my thinking has flipped on the `data-object`. I was getting all balled up over ownership, security and shareability. The bottom line is, I think we had it right before. We don't need "public" and "private" data-objects. By definition, all data-objects are private.

It's important to remember that we solved (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5223/referencing-standard-dbs-and-versioning?postid=5247#5247) the "version-reference" problem (in part) by making data-objects immutable and permanently tied to a single `slice`. This allowed us to place versioning at the slice-level and therefore guarantee a consistent set of syncable objects.

If a reference version changes for a slice, it's the PIM's responsibility to update all data-objects in the set as needed to reflect those changes (using data-object adds/deletes).

But the problems come when we try to "re-share" a data-object. How is that done when a data-object is tied to a single slice? Also, how do we share something that is inherently private (and so un-sharable)?

This is where I was stuck. My first thought was to simply share slices. But is that granular enough? Would that violate privacy? I believe what is really needed is to re-create a slice that is specific for the receiver-group and then keep it updated as required. That falls outside of Sandpiper's responsibility.

If you have the requirement of sending data, you become a data-publisher (i.e. Sandpiper server) with all of those responsibilities and support needs.

privacy (/spaces/127/sandpiper/searchresults?keyword=%22privacy%22&searchtags=1)

Like

Reply (/forums/post?tid=5128&ReplyPostID=5364&SpaceID=127) Not answer