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

The Slice

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

Content is (and will be) moved around the aftermarket in discrete defined bundles. Something we all hear on a regular basis is: "*Send me ACES, PIES and Assets for <fill in the blank>*". The person uttering those words and the person hearing those words already have a mutual understanding of what goes in the blank. Maybe it's a new line of alternators, or a private label filter program. It could be an existing product line and a new customer relationship. It was defined and negotiated long before the Content people got involved, and it was defined at a higher level of decision-making than the Content people. It was negotiated by the suits in the boardroom. For one party, the negotiation ended in a finite list of part numbers and a promise to add more in the future. For the other party, the negotiation ended with a promise to do something valuable with the data every time a fresh bundle is sent. Only the parties involved in the negotiation know what should be in the bundle. We'll call that bundle a "slice". The author (sender) of the slice keeps track of what apps, parts and assets should be added or removed over time. The receiver of the slice is generally aware of the approximate quantity of objects to expect, but trusts to author to curate the slice correctly. To the sender, the slice has these elements:

1) A selection criteria for the apps, parts and assets to export from the source PIM system

2) A receiver that is expecting to receive Content

3) A Procedure for sending the Content - This would include things like FTP credentials, expected frequency of sends, contact info for the humans to notify when the content is sent.

To the receiver, the slice has these elements:

1) A sender who has agreed to send Content

2) A selection criteria for what to replace or update in the destination PIM system

3) A Procedure for consuming the Content - This would include expected frequency of sends, contact info for the humans to talk to when there is a problem, the people internally to route the Content to, the assessment tools and processes used to validate the Content

For all intents and purposes, the slice is a contract. In the current ecosystem, the slice is generally implied but not actually well documented. We see portions of one side of it written in vendor agreements. The agreement may say something approximating: "vendor agrees to send catalog data no less that monthly....". This nebulous "agreement" is not actionable by a machine (which is the purpose of Sandpiper)

What we must do here is define a structure (xml?) for both side's elements to codified and *possibly* transmitted across the API channel. Some concept of cryptographic signing (not encryption) may be appropriate. There may not actually be a need to transmit the explicit elements of the slice - much like the elements of today's bundles are mostly held in institutional memory and force of habit.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5112&SpaceID=127)

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

The starting point is that both parties use the same string of characters to refer to the slice. This string of characters is the "slice id". The slice id does not need to be unique across the aftermarket - it just needs to be unique in the contexts of the sender's and receiver's systems. It could be something as short and simple as what we call a "line code" today (ex: "AB1"). Using a longer and random-ish string would probably be wise in order to lessen the chance of a collision. It could even be the same format as the oid (14 characters starting with the ACA brand id).

Like

Reply (/forums/post?tid=5111&ReplyPostID=5113&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
8/22/2019

This sounds good!

The declaration and definition of the slice, since it belongs to the agreement, should be in the agreement itself, to keep the API super simple -- you'd only ever act on the slice ID, which could even represent a subslice within (that also gets its own unique ID) agnostic of what that actually means to both parties. That way the API can purely be, "Give me what you have for this universe: 123456789", and both know exactly what that is supposed to mean, yet the program itself only knows that it fetches from a unique universe ID, and compares against that universe ID in the canonical pool.

However, for simple receivers, the top-level slice ID would be the only point they'd probably want to reference, so we need to keep that available for the "full replacement" mentality.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5114&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/22/2019

To me, a lot of what you've described for a slice feels out of scope (at least for a 1.0 version). For example, do we really need to store internal-pim "selection criteria", or document "the assessment tools and processes used to validate the Content"?

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

Like

Reply (/forums/post?tid=5111&ReplyPostID=5116&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/22/2019

Can you give more detail on the "subslice" requirement? It seems to me you could just define a new "slice" if you need to deliver a separate set of data.

Here is what I currently have for **Slice** (and **DataObject**) models:

Model	Data Type	Description
Slice	Object	Defines a set of product objects
id	string: uuid	
name	string	unique name assigned to the slice
publisher_id	string: uuid	trading partner that owns the slice
subscriber_id	string: uuid	trading partner assigned to receive content for this slice
update_frequency	number	how often might the slice be updated (in minutes)
content_hash	string	crc representing all object ids in the set
content_count	number	number of DataObjects in the slice
last_update	date-time	when the hash and count were updated
last_sync	date-time	when the receiver completed a sync
DataObject	Object	
id	string	
owner_id	string: uuid	unique id of creator
payload_type	enum	["aces", "partspro", ...]
payload	string	

SliceContents, not shown here, represents the many-to-many relationship between Slice and DataObject (assuming you might want to send the same DataObject to more than one receiver). This SliceContents might be a big challenge for internal-pims to maintain.

Right now I have a **Publisher** (that owns the primary data pool) and a **Subscriber** (that consumes the primary data pool). One thing I'm considering is a **TradingPartner** (which could be a publisher, subscriber or both) and speaks directly to the two-wayness question.

slice-data-model (/spaces/127/sandpiper/searchresults?keyword=%22slice-data-model%22&searchtags=1)

Like

Reply (/forums/post?tid=5111&ReplyPostID=5117&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

It may be out of scope, but if it's not defined somewhere it devolves down the equivalent of "zz" (mutually-defined) in EDI. When things are not defined, it makes interoperability between disparate systems less likely. One other reason to store all that "private" minutia (that has no business crossing the API pipe) is for crypto signing. If both sides digitally sign a chunk of data that spells out the internal specifics of the partner's organization, you get something that closer approximates a binding marriage.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5118&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

On the subject of "SliceContents" being hard to maintain:

My present-day approach to what currently amounts to SliceContents is pile of php scripts that are specific to each receiver. (This is on my home-grown PIM system that lives on a LAMP stack.) Each script contains the list of internal part type id's (not ACA's partterminology) that the objects are tagged with. For instance all the ACES apps for a given coated rotor are tagged with a different part typeid than the apps for the non-coated equivalent. At the moment I run the script, I get an ACES & PIES xml dump to send to the receiver. I'm not maintaining a many-to-many table of object-receiver connections. I suspect I'm not alone in this approach.

If I did use a many-to-many mid-table to connect receivers and individual data objects, I'd need a house-keeping process to audit for orphan objects that my product managers had forgotten to assign to a receiver.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5121&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/22/2019

On the subject of sub-slice:

My opinion is that a slice can't be sub-divided. It is the molecular unit and the content objects are its constituent atoms. I think we would be trying to satisfy an insatiable appetite of sub, sub-sub and sub-sub-sub division. In think instead we go the other direction and make a "plan" or "contract" or "agreement" contain a collection of slices (each identified by uuid). Each slice in the agreement would contain an arbitrary number of objects and carry the hash of id's.

The author's system would break up the agreed-upon list of objects in the plan into something it could reliably repeat (makeid, partterminologyid, part number, modification date etc). Grouping by modification week or month would be most ideal because it would cram all the recently-modified objects into a slice or two. The rest of the slices would hash to the same result as a previous sync. Slice hashing starts to look very interesting when there are dozens of slices instead

of one huge one. Side-note on hashes: I think md5 would be a better choice for a hash than crc. It has way better collision resistance and is probably easier to implement in the languages that Sandpiper would likely be build with.

In order for the receiver to calculate a hash, it would have to "keep the wrapper" so-to-speak from the last sync. It would have to store its most recent hash for a slice locally in order to detect a change. The other possibility would be for the receiver to keep the sliceid locally as an attribute of the data object in the secondary pool. The hash could be re-computed on the fly for the slice by querying the pool by sliceid and hashing the resulting oid's.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5122&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/22/2019

> "Grouping by modification week or month would be most ideal because it would cram all the recently-modified objects into a slice or two. The rest of the slices would hash to the same result as a previous sync. Slice hashing starts to look very interesting when there are dozens of slices instead of one huge one."

Sounds a lot like the way Mac TimeMachine backups work. They break files into "Bands" to allow finer-grained versioning.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5123&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
8/23/2019

> **On 8/22/2019 10:33 PM, Luke Smith said:**
>
> My opinion is that a slice can't be sub-divided. It is the molecular unit and the content objects are its constituent atoms. I think we would be trying to satisfy an insatiable appetite of sub, sub-sub and sub-sub-sub division

I can see this and think you and Doug may be right, that it's too complex to do in this round. I have been hoping that I could do this but it might be too difficult.

Like Doug mentioned about the time machine backups, my thought had been that there is only "Slice", but that each slice could refer to a subset of records within a slice. But like you are saying, perhaps it should be one level only, and if you want to define a deeper division, you do so with multiple slices when setting up the pool and leave it at that.

Then perhaps the node software itself could help the user establish some groupings and enforce consistency, even though the output is one level.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5124&SpaceID=127) Answer

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

I like this reasoning. I was thinking we might need a "Subscriptions" level to hold all Slices for a Subscription:

```
Subscriber -> Subscriptions -> Slices
```

But I don't really see much need for the Subscriptions level. A Subscriber (on that server) can have lots of Slices (from that publisher). Each Slice will have a name, description, start_date, end_date, and content_hash. I'm not sure why we'd need to group them higher than that.

I see Slice names and descriptions like:

```
"BP-Epi", "DRiV: Epicor ACES Brake Pads"
"BP-WHI", "DRiV: WHI ACES Brake Pads"
```

Note that one Data Object might belong to both of these Slices.

The concept of SliceBands, if we were to implement that, would go below Slices and be invisible to the UI. It would just be an optimization (which is probably why we shouldn't design it until we see we have a performance problem).

Like

Reply (/forums/post?tid=5111&ReplyPostID=5125&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
8/23/2019

> **On 8/23/2019 9:14 AM, Doug Winsby said:**
>
> But I don't really see much need for the Subscriptions level. A Subscriber (on that server) can have lots of Slices (from that publisher). Each Slice will have a name, description, start_date, end_date, and content_hash. I'm not sure why we'd need to group them higher than that.

The subscriptions element I think is good, but I do think it'll serve more as a "Here's what I plan to get from you, on this schedule" connection rather than "Here's how I plan to use what I get from you". No classification or grouping at that level.

> **On 8/23/2019 9:14 AM, Doug Winsby said:**
>
> The concept of SliceBands, if we were to implement that, would go below Slices and be invisible to the UI. It would just be an optimization (which is probably why we shouldn't design it until we see we have a performance problem).

I agree there. My thought was originally to have each slice have a "SubSlices" element, that would contain another set of Slice elements. That way you could query the XML and just say "Get all of this particular value, wherever it's used on a slice element, starting here". Each subslice would be a division within the parent, a way to access the finer grains. Perhaps we can reserve the element and reconsider in Dev 2.0 -- even if just to put it on the map.

I also was thinking that we could say that the API could always use part number as a slicing agent -- the one thing that should always be universal among product data is some kind of pivot around the part number as a unique identifier. We can specify that this only works if the part number is a single string, a single value, to be that same unique ID approach. Then the API could offer to granulate down one additional level. Perhaps that's something to table though.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5127&SpaceID=127) Answer

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

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson) You and I were thinking along the same lines with the Part Number as a key. See The DataObject (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5128/the-dataobject) discussion for some of my thoughts.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5130&SpaceID=127) Answer

Luke Smith (https://autocare.communifire.com/people/autopartsource)
8/23/2019

One thing to consider is that the part number may mean different things in different agreements. The same part number can be described, priced and applicated differently depending on who's asking - case-in-point: "linecodes" in today's ecosystem. So a part-keyed API endpoint is likely not granular enough.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5131&SpaceID=127) Answer

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

I understand a part number might use different numbering (naming?) depending on the customer. I was suggesting a way to lift the cover a bit on the black box. So this would be a key into the payload (using the part number found in the payload).

Like

Reply (/forums/post?tid=5111&ReplyPostID=5132&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
8/26/2019

The PIM-to-Sandpiper interface is certainly out-of-scope, but it makes sense to think it through to make sure we're not building something that can't work in real-life scenarios.

> "For instance all the ACES apps for a given coated rotor are tagged with a different [internal] part typeid than the apps for the non-coated equivalent. At the moment I run the script, I get an ACES & PIES xml dump to send to the receiver. I'm not maintaining a many-to-many table of object-receiver connections."

(I added "[internal]" to your quote here for clarity.)

Even with this type of internal grouping, though, you'll need some way to determine which data pool object-ids are the same and which ones need to be added (what's left over needs to be dropped from the pool).

That could be implemented by adding Sandpiper object-ids to the respective tables in your PIM (possibly at a part group level) or by having an external table that links object-id to PIM primary ids. A PIM implementation would probably always work at the same level of DataObject granularity to make this all work.

My data models have been at the API level so far. I think it might be helpful to create a database model (ER diagram) as well.

Like

Reply (/forums/post?tid=5111&ReplyPostID=5140&SpaceID=127) Answer