



Sandpiper (/spaces/127/sandpiper) ▸ Discussions (...)

Private Space · Technology Standards (/spaces/9/technology-standards/technologyhome)

(/spaces/127/sandpiper)

Activity Stream (/spaces/127/sandpiper/?act=1)

People (/spaces/127/sandpiper/people)

Content ▼

Posted in: General (/spaces/127/sandpiper/forums/4997/general)

Project Requirements

✉ Unsubscribe from this discussion

📡 Subscribe to RSS (../..../..../spaces-cf/forums/rss-space-posts.ashx?

spaceID=127&topicID=4998&forumID=4997&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/12/2019

(From an email I sent to Luke and Krister on 7/27/2019)

I don't have the benefit of some of your previous discussions (besides what's been presented on the phone calls), but I thought I'd share some of my current thoughts to see if any of it comes close to the approach you have in mind.

Fundamentally, I see **everything driven from the product side**. Everything (including application data) is just an attribute of the product.

1. Each "Marketed Product" has a unique id ("pid")
2. "Product Data" is a set of key/value pairs for each Marketed Product
3. The goal is to sync two trading partners' Product Data (as often as required to stay in sync). Receiver data is immutable which simplifies the sync.
4. A "Product Segment" is the lowest level API query/update for a set of Product Data (which defines the operation granularity). These would not be "ACES", "PIES", but rather "Make/Model/Year/PartType", "Item", "Description", "Pricing", "EXPI/CTO"). Notice that these segments do not use existing PIES "Keys". For example, the lowest level for a Description is all Descriptions for a "pid" (in a JSON collection)

5. Trading Partner "Product Data Hierarchy" is segmented as: Brand (BrandAAIAID) / Product Group (PartTerminologyID) / Product Segment.
6. The current state of Trading Partner Product Data is defined by a set of version codes (stored at each level of the Product Data Hierarchy). These codes are assigned by the supplier (not calculated by the receiver) and probably implemented as a data hash of all data included in that level.

The sync method would be to ask a Trading Partner for the highest level version code. If it matches, they are in sync. If not, walk down the Hierarchy asking for version codes to see which Product Groups must be synced. Now that you have that list, ask for UUIDs for each Product Segment and then update as required.

This sync method may produce too many UUIDs to check (I'd think anything under 1000 is fine). We'd need to see if other levels in the Hierarchy might make sense to limit the number. Desired "Language" is another complication which should probably be in each query/update.

We would not use the actual ACES and PIES xml for any of this. For this reason, I think we should use JSON rather than XML for the API. It seems to be more compact and better supported.

Suppliers would still produce ACES and PIES xml for legacy delivery, and could also be used as a source to create the sync database to be used as explained above.

This seems a bit more complicated than what was in my head. But isn't that always the case? :)

```
email-history (/spaces/127/sandpiper/searchresults?keyword=%22email-history%22&searchtags=1)
```

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=4999&SpaceID=127)



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)

8/12/2019

Here's a snapshot of my most recent ponderings:

Method Name	Description	Parameters	Response

originOut	Accept a sliceID and list of objects in the branch and give back object data and drop advice to client. Server is the authoritative source of data for the slice. Client is the branch.	sliceID, OID list currently in client's pool	Object Data that client should add, OID List that client should drop.
branchIn	Accept sliceID, object data and drop list from client. Client is the authoritative source of data for the slice. Server is the branch.	sliceID, Object data that the server should add to its system.	Simple acknowledgement
listSliceObjects	Give client a list of OID's the the server currently has for the slice. The server can be Origin or Branch.	sliceID	OID list

Scenario 1: Manufacturer hosts public Sandpiper service and retailer periodically (daily) connects to check for and sync adds and drops. API represents the manufacturer's origin pool.

Retailer posts sliceID and list of current OID's to /originOut

Manufacturer responds with object data and droplist of OID's

Scenario 2: Retailer hosts public Sandpiper service and manufacturer periodically (daily) connects to sync adds and drops. API represents the retailer's branch pool

Manufacturer posts sliceID to /listSliceObjects

Retailer responds with list of OID's in the branch pool

-- new connection is made seconds or minutes later---

Manufacturer posts sliceID, object data and object droplist to /branchIn based on the list of OID's reported moments earlier in the previous transaction

Scenario 3: Un-sophisticated E-cat hosts public Sandpiper service and manufacturer periodically (monthly) connects to push "full" datasets. API represents the E-cat's branch pool

Manufacturer posts sliceID and all object data to /branchIn

Scenario 4: Manufacturer hosts public Sandpiper service and un-sophisticated E-cat periodically (monthly) connects to pull "full" datasets. API represents the manufacturer's origin pool.

E-cat posts sliceID to /originOut All objects in the slice are contained in the response.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5000&SpaceID=127)

Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)

8/12/2019



(Email I sent on 7/29)

Some good info, I'll share my notes on it too.

In order to start a synchronization, we need both actors to agree to do it. One is the origin, and the other is the branch – one is the canonical source and the other is downstream of it. And they need to know what their role is ahead of time, for example you subscribe to a manufacturer's product data as a branch. This doesn't mean you can't turn around and be the origin for other systems further downstream (as long as you have permission), but each interaction has upstream and downstream. And you should know what data you're going to talk about before you begin, so that invalid interactions can be refused and the programming can be tightly scoped.

So before they can enter into an API interaction, both actors sign a contract with one another, and become officially associated.

The Contract

The contract establishes the rules for the interaction and the expected outcomes. This is an XML file that details who, what, when, where, and how ("Why" is too existential), and the actors hand it back and forth to each other to signify explicit agreement with its content and the context it establishes. This also makes programming nicer in some ways, since there's no state of the connection to track, just the document itself however it arrives.

The contract is also where we establish and subscribe to the product sets that are available (I've been calling them slices) and link them to other systems like the PCdb, proprietary internal classifications, etc.

The content:

- 1. Who: the basic details of the origin and the branch (including contact information, individuals, links to branding standards like NAPA and Autocare)
- 2. What: Available product sets for this association, with a unique ID for each
- 3. When: Subscriptions to product sets that the branch wants to carry, with a unique ID for each
- 4. Where: Product sets and subscriptions carry URIs through which an actor can fetch or push information, or beginning an API interaction
- 5. How: Origin and branch specify their capabilities for synchronizing information and their preferences, credentials for the set & subscription URIs

Types of Synchronization

In the case of the full replacement model, the product data is synchronized in full using references in the contract manifest (no awareness of branch state is required, so the stateless context of the contract is enough). In the case of the API model, having completed the association step, the two actors enter into a "rich" synchronization using the unique IDs in the API. That's where the fun starts for the future stuff..

I agree with you both that unique IDs should be the only method of sync for the API interaction. And the only types of changes should be add or delete – not "Modify", the most evil thing invented by data demons.

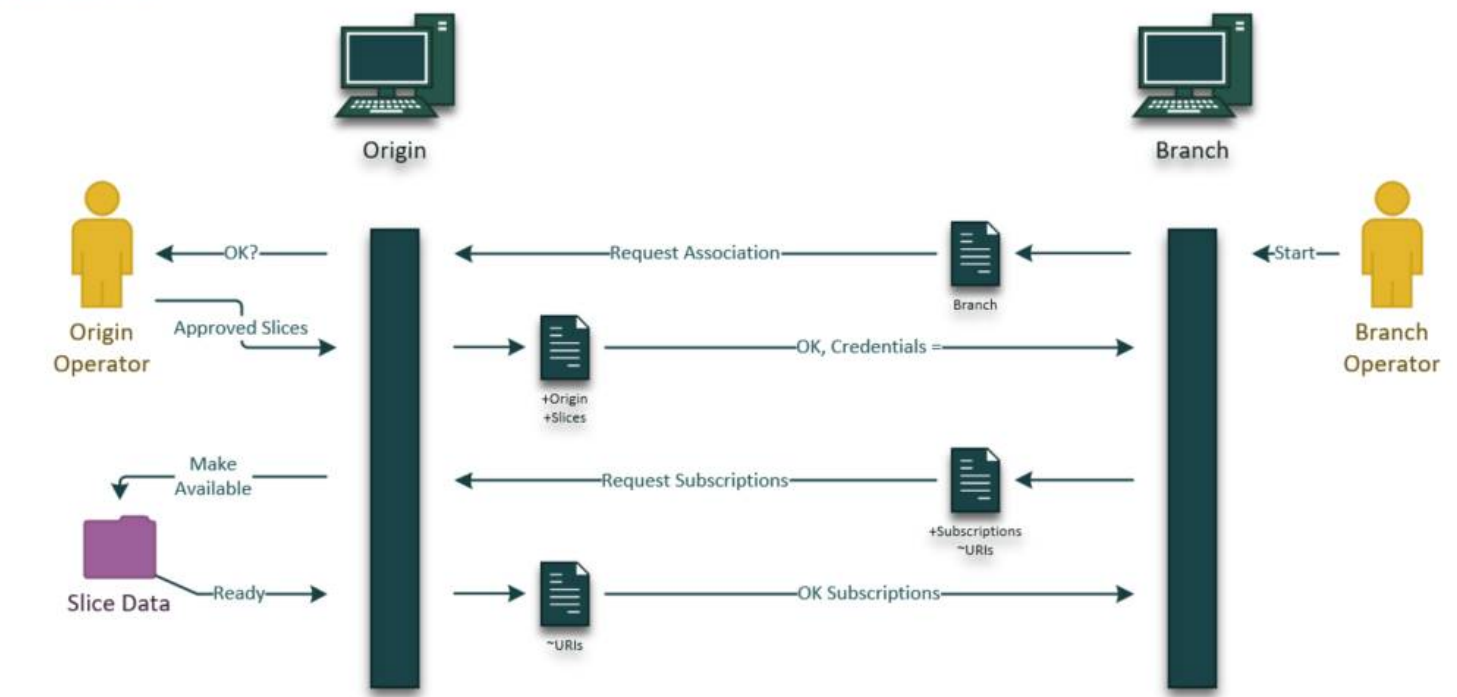
Extra Credit

Here's an interaction drawing I did of how I thought the initial negotiation and association might go:

Actor	Action	Contract XML

Branch	Operator requests association with the origin	Init, branch details completed
Origin	Operator approves the branch for specific slices of data	Revised with origin details and slices completed
Branch	Requests subscriptions to some or all slices	Revised with subscription and delivery preference, URIs
Origin	Make slice data available	
Origin	Return understanding of subscriptions via preferred delivery method	Revised with URIs

Association



👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5001&SpaceID=127) Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>) 8/12/2019

(From email to Luke on 8/8/2019)

I'm coming around to some of the concepts you and Krister have presented, but it was fundamentally different from what I'd envisioned originally. I want to drink the Kool-aid, but I'm still a bit skeptical, I guess.

Here are some thoughts from our lead architect as he understands the project:

"It sounds like what they're looking for is to exchange content agnostic bundles of data in a way that can be reliably tracked, supporting syncing between a data creator and a receiver."

"A bundle would be identified by an object id and probably contain a header describing the data and any information necessary to process the data (VCdb version in the case of ACES). It may refer to a particular external "contract" that links bundles together as a set. A contract might consist of a set of all bundles that would currently constitute a file in ACES or PIES. A contract might specify allowable content."

"In the case of ACES, it would probably require that a bundle contain all applications for a BaseVehicle and Part Terminology ID. Ideally, the receiver should be allowed to define what they need in terms of content to process the content in the most efficient manner. Of course, the more flexibility given to receiver to define expected content, the more work it is for the creator to satisfy multiple receivers."

"Receivers will need to do a drop and replace in their internal system based on the content of a bundle. To do that, they will most likely need a table linking the bundle object id to internal ids representing the content. Creator would also need a link between internal ids and object ids in order to build the bundles. The needs in this respect would likely vary significantly from one creator to another and from one receiver to another."

Before we even get that far, though, I'm wondering if we should step back a second and better understand the problem.

I know the goal is faster releases of information to the end-user. But without a direct pipeline from supplier to end-user, that might not be under our control. **Many of the delays, I suspect, are due to data quality issues. Especially with regards to ACES data.** Receivers will still want to validate, check for over-mapping and overlaps, and then provide a feedback loop for corrections.

And ACES has the particularly nasty problem of being tied to specific VCdb releases. PIES has now also created a dependency on PCdb version for code tables which concerns me some (allowing deletes of existing Codes and changes to business rules based on PCdb release date, for example).

I understand your concept is not to worry about the payload. But if we don't at least consider how a receiver might handle these payloads, we might be building a system that is unusable by the receiver.

Maybe it would be a faster win if we limit the project to PIES-only data to start. That would be much easier to validate and certainly easier to manage. I was thinking of a more traditional REST API, one-way from the supplier, but with the ability to initiate an update session with a receiver.

Maybe something like the following (using simple date logic for sync processing):

/segments

/segments/{segid}/lastsync

/segments/{segid}/items

/items/{pid}/descriptions

/items/{pid}/pricesheets

/items/{pid}/prices/{pricesheetid}

/items/{pid}/packages

/items/{pid}/assets

I'm not trying to throw cold water on any valid approach. I just want to make sure we've considered all options.

One thing I failed to mention with this (simplified) approach is that it would mean developing a derived PIES schema because we'd use UUIDs and deliver groups of data that would not be valid xml against the current xsd. (Or, my preference, JSON using just the concepts, codes, etc. in PIES).

I would not want to feel constrained by past mistakes or require support for old PIES versions.

I know this would mean parallel efforts when making PIES changes in the future, but I think it would be worth it (maybe they could eventually merge).

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5002&SpacelD=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)

8/12/2019



(From email to Doug on 8/8/2019)

I appreciate your perspective - thanks for putting the effort into typing all that out.

The fundamental thing we need to solve is the need for "full" data integrations on every cycle. I agree with your assessment on receivers needing quality control and the ability for feedback. The resulting outrageous volume of validation man-hours in the current paradigm is what is holding

us back. There's no way to approach "real-time" when Epicor, WHI, etc are essentially doing a full re-integration every month. Adding granular end-to-end object identification to our PIM ecosystem allows the origin system/author to closely couple as a trusted extension of receiver.

No matter what shape our solution takes, it must provide granular object identification. Otherwise, we will just be inventing a faster and fancier version of the full-replacement ecosystem we have today. We must have a reliable way to tell receivers to surgically remove specific bad content and implant specific new content while keeping the humans out of the process.

Granular object identification is my main focus at this point. It is the only hill I'm willing to die on here. The level of granularity isn't even all that critical. The magic is that when the origin PIM makes a substantive change to an object, it applies a fresh UUID to the object. There's no concept of "change" - just adds and drops conveyed down-stream. The down-stream system either has the object or not - there is no "change"

(Follow-up email to Doug on 8/11/2019)

Doug, I've been thinking more on the difference-of-approach that you and I have. I believe in a perfect world, your REST API that is always open for business to be the authoritative source for content would be the bee's knees. There would never be a divergent copy down-stream because there would only be one single place to ask about a specific brand: the author's sandpiper endpoint. There would never be a need to convey an ACES or PIES file to a trading partner - just a URI and creds. There would not be any need to tell the down-stream to add, drop or change anything because there would be zero non-authoritative repositories out there.

The universal object focus of mine is an acknowledgement that there will always be cloned repositories in the wild. I don't see us ever getting away from that fact. Case-in-point: Epicor could not guarantee a good and consistent UX to their LaserCat users if there was no local repo or Epicor-controlled cloud repo. Relying on dozens of manufacturers to stand-up identical, snappy and reliable API's is not realistic. No "Receiver" in our ecosystem would put themselves in that precarious position.

Krister made a point in our call last week that I want to reiterate. The origin/branch-object-conveyance model is about synchronizing pools of data - much like bitcoin wallets. My authoritative pool (my PIM) is not syncing with Epicor as a whole. My pool is syncing with a pool in Epicor's lobby (so to speak). They can do as they see fit with that up-to-date wallet. That branch pool in Epicor's lobby could be asked at any moment to export ACES & PIES files, and it would dump files that agree exactly with the authoring PIM's pool. They are still free to take 90 days to process the output of the pool if their business model requires that.

There may be room for both approaches side-by-side in sandpiper. I could see an e-commerce platform doing the kind of queries you described to dynamically retrieve PIES-ish data from the manufacturer when a product page is loaded

👍 Like



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/12/2019

(From Email to Luke on 8/11/2019)

I think we generally agree. I wasn't expecting the API to be used directly for updates to Lasercat (or any other back-end solution). I always expected there to be a need for a sync procedure between trading partners.

In our internal discussions here, we've been calling these "lobby" databases "staging" databases to indicate they're simply a starting point for the receiver (but always up-to-date). This concept seems similar to yours, with the supplier having the authoritative (source) data, and each receiver having a synced copy (staging database). The main difference, I think, is that they **could not reverse roles**. It's only one-way (supplier to receiver).

In the reference implementation we're thinking about, both the source and staging databases would use mongodb* and would contain **all partner traded data** (with a receiver having many sets of supplier data). The receiver could poll each supplier on a regular basis to see if anything is new, or even better, the **supplier could initiate a sync** ("hey, I have new data, come get it").

On the supplier side, all data in a "segment" (the smallest syncable unit) must use the same "Standard version" (ACES 3.2, PIES 7.0, etc.) and the same "reference data version" (vcdb release, pcdb release, qdb release padb release). These versions are stored in the segment, and a segment is specific to one trading partner relationship.

Every deliverable piece of data is identified by a **uuid** and is never removed (just maintained with a status code and status date). I suppose a "clean up process" could eventually remove deleted data once it has been synced.

I guess the biggest conceptual differences are:

1. **Two-way sync**. This adds a lot of complexity and might not really be necessary given the supplier is the author of the data. We could develop a separate feedback API for handling validation errors or other data problems.

2. **Blackbox payloads**. Won't the receiver need to know what's in the payload to be able to interpret it? We have a unique opportunity to do things right (including the standardization of "new part introduction"). If we just package up current xml, it really doesn't improve the process (only the delivery).

**MongoDB is an open source, NoSQL database that provides support for JSON-styled, document-oriented storage systems. It supports a flexible data model that enables you to store data of any structure, and provides a rich set of features, including full index support, sharding, and replication. If scalability is a concern, there are hosting plans available on AWS, Google and Azure.*

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5004&SpaceID=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)



8/12/2019

(from an email to Doug on 8/11/2019)

On the subject of black-box payloads: My vision is more like translucent-box. The object (or segment) would be attributed with a typeid that would imply parsing rules and expectations of the receiver. ACESapp, PartsProApp, TechdocApp, PIESitem, etc. Some would have an strictly-defined schema and xsd, some would be more don't-ask-don't-tell. IP knows that its payload is TCP, UDP, IGMP, ICMP, proprietary, etc., but does not concern itself with the business rules of that protocols - it simply keeps track so the next layer up the stack can handle the payload correctly.

On the subject of two-wayness: There's no guarantee that the server is representing the origin pool. Generally speaking I would expect it to. Consider however the case of a large receiver like Napa, zone or opticat. They would not trust a thousand small suppliers to stand up reliable API services. Those large receivers would insist on hosting, and the origin pools would connect at their convince and push content into the API. Thats the rationale for the protocol going both ways - it's to avoid having to build two different server flavors (origin server and branch server) and two different client flavors (origin client and branch client)

(from an email to Doug on 8/12/2019)

Here's my artist's concept of the originOut method:

originOut - Accept a sliceID and list of objects in the branch and give back object data and drop advice to client. Server is the authoritative source of data for the slice. Client is the branch.

Parameters

- sliceID
- OID list

- TOTP password

Response

- Object set data
- Drop list

Request xml example (http POSTed to the API's /originOut endpoint)

```
<sandpiper env="prod" version="1" partner="ACME Brakes" TOTPAuth="402183" sliceid="AB1">  
  <o>DHCN3d4874g982</o>  
  <o>DHCN221t23480y</o>  
  <o>DHCNe768223g73</o>  
  <o>DHCNp4f9120338</o>  
  <o>DHCNx4fd1201qs</o>  
  <o>DHCHRvf112p33Q</o>  
</sandpiper>
```

Response xml example

```
<addObjects>  
  <object type="ACESapp" id="GQBF3d4874g982">  
    <App action="A" id="1" ref="12646417">  
      <BaseVehicle id="5888"/>  
      <Qty>1</Qty>  
      <PartType id="6832"/>  
      <MfrLabel>Particulate</MfrLabel>  
      <Position id="1"/>  
      <Part>3013</Part>  
    </App>  
  </object>  
  <object type="ACESapp" id="GQBF221t23480y">  
    <App action="A" id="1489" ref="13515778">
```

```
<BaseVehicle id="130977"/>

<SubModel id="96"/>

<EngineBase id="2068"/>

<CylinderHeadType id="6"/>

<Region id="3"/>

<Qty>1</Qty>

<PartType id="6832"/>

<MfrLabel>Particulate</MfrLabel>

<Position id="1"/>

<Part>3693</Part>

</App>

</object>
```

```
<object type="PIESitem" id="GQBFx4fd1201qs">

  <Item MaintenanceType="A">

    <ItemLevelGTIN GTINQualifier="UP">00734776312363</ItemLevelGTIN>

    <PartNumber>3693</PartNumber>

    <BrandAAIAID>DHCN</BrandAAIAID>

    <MinimumOrderQuantity UOM="EA">1</MinimumOrderQuantity>

    <PartTerminologyID>6832</PartTerminologyID>

    ...

  </Item>

</object>
```

```
  <object type="DigitalAsset" id="GQBFRvf112p33Q">
<assetData>RG8geW91IGtub3cgd2hhdCB0aGF0IHNVdW5kIGlzLCBlaWdobmVzcz8gVGhvc2UgYXJlIHRob2ZSBTaHJpZWtpb
mcgRWVscy4gSWYgeW91IGRvbid0IGJlbGlldmUgbWUslGp1c3Qgd2FpdC4gVGhleSBhbHdheXMgZ3JvdYBsb3VkZXlkd2hl
biB0aGV5J3JlIGFib3V0IHRob2ZlZWQgb24gaHVtYW4gZmxlc2guElmIHlvdSBzd2ltIGJhY2sgbm93LCBJIHB5b21pc2UsIG5vI
Ghhcm0gd2lsbCBjb21lIHRob2ZlvdS4gSSBkb3VidCB5b3Ugd2lsbCBnZXQgc3VjaCBhbiBvZmZlciBmcm9tIHRob2ZSBFZWxzLg==
</assetData>

  </object>

</addObjects>

<dropObjects>

<o>DHCNeu6f2l3g73</o>
```

<o>DHCNx4fd1201qs</o>

</dropObjects>

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5005&SpaceID=127) Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>) 8/13/2019

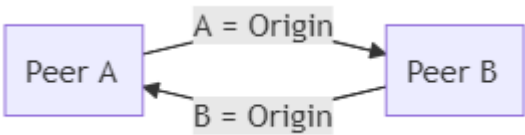
I'll have more thoughts as I catch up, but as far as the concept of two-way-ability: I completely agree that it's not feasible. When I've talked about two way abilities, I mean that the software itself is capable of being either an origin *or* a receiver, but in any given slice of a relationship, it can never be both.

In the document I wrote up, I mention this:

Origin and Branch

In an exchange, there can be only two peers at once: an *origin* and a *branch*. The origin is the holder of the most authoritative version of the data, and the branch is a consumer of that data.

The origin is assumed in all interactions to be the canonical copy of the information being exchanged. However, every peer needs to be capable of acting as either an origin or a branch; sometimes the roles will be reversed. Their primary commitment is to resolution of the truth, and the holder of that truth is tied to the data set, not the relationship.



However, the drawing I did was maybe confusing, it does seem to imply that there's a two-way simultaneous relationship. I'll revise it..

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5006&SpaceID=127) Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)

8/13/2019



Ah, also, I think I should add some language that makes it more clear that this is an either-or thing. I'll also do that.



Reply (/forums/post?tid=4998&ReplyPostID=5007&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)

8/13/2019



"every peer needs to be capable of acting as either an origin or a branch; sometimes the roles will be reversed."

I think a real-world example would help me understand this requirement better. For example, when would NAPA need to update information on Tenneco's (authoritative) data pool?

two-wayness (/spaces/127/sandpiper/searchresults?keyword=%22two-wayness%22&searchtags=1)



Reply (/forums/post?tid=4998&ReplyPostID=5008&SpaceID=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)

8/13/2019



NAPA would not update Tenneco's pool, but NAPA would be the authority for other product data (not Tenneco). It would need to update other branch pools that have nothing to do with Tenneco or Tenneco's products. The point is that NAPA's sandpiper implementation would need to handle both roles. Someone like myself (only functioning as an origin) could leave the "branchIn" method un-implemented in their sandpiper API.



Reply (/forums/post?tid=4998&ReplyPostID=5009&SpaceID=127)

Answer

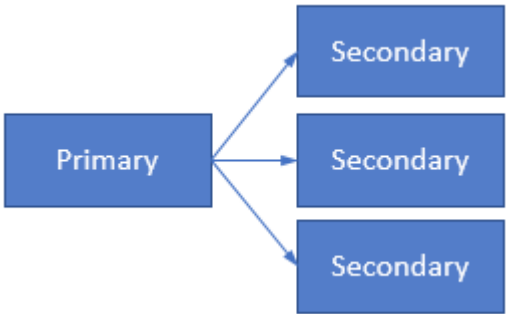


Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/13/2019

Maybe the requirement is really "Primary-Secondary" rather than "Peer to Peer". Something like this:



Tenneco could be "Primary" for relationships with NAPA, Epicor, etc. NAPA could be "Primary" for its stores (for example). But the data pool is only one or the other. If you want to be both Primary and Secondary users of Sandpiper, you need two separate data pools (with a single role).

With this approach, the API is much cleaner and easier to implement.

[two-wayness \(/spaces/127/sandpiper/searchresults?keyword=%22two-wayness%22&searchtags=1\)](/spaces/127/sandpiper/searchresults?keyword=%22two-wayness%22&searchtags=1)



Like

[Reply \(/forums/post?tid=4998&ReplyPostID=5010&SpaceID=127\)](/forums/post?tid=4998&ReplyPostID=5010&SpaceID=127) Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



8/13/2019

That's exactly it. It's peer-to-peer only in the sense that the actors are both peers on the network, both sovereign citizens of the standard. The data itself, and the agreement itself, are origin-to-branch, primary-to-secondary, authority-to-derivative.

I would suggest that at least a stub API method be present for any implementation, and we specify an error message that says basically "Hey, I can't receive" when an attempt is made.

I've revised the drawing and added some info to the documentation:

Peers and Their Roles

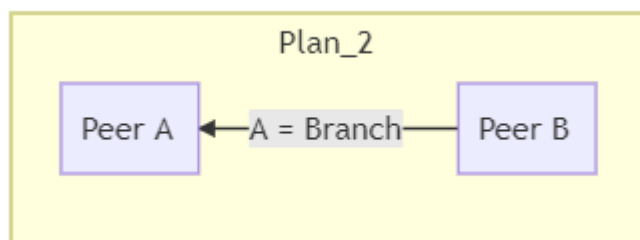
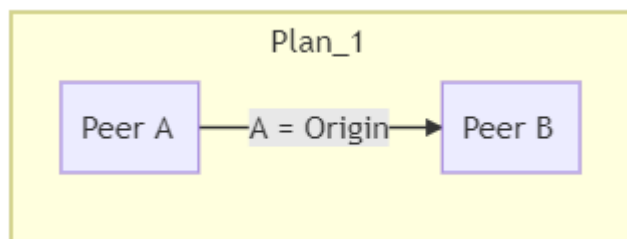
Individual nodes that can take part in a Sandpiper-driven exchange are known as *peers*. However, within an exchange, they take on a primary or secondary role.

Origin and Branch

In an exchange, there can be only two peers at once: an *origin* and a *branch*. The origin is the holder of the most authoritative version of the data, and the branch is a consumer of that data.

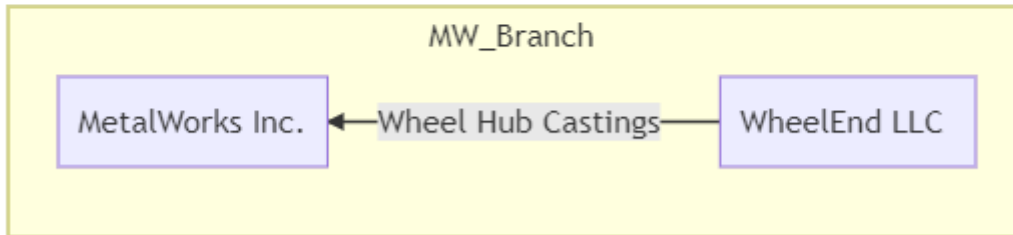
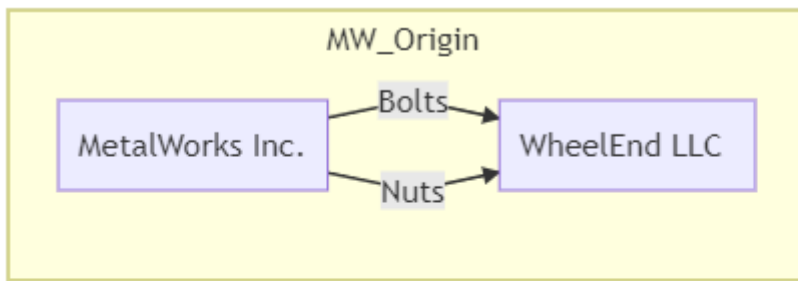
For an agreement on any given set of data, the origin will always be the origin, and the branch will always be the branch. This is established in level 1 as part of the plan, and cannot change without a new agreement.

Every peer should be *capable* of acting as either an origin or a branch; sometimes the roles will be reversed, depending on the plan and the business need.



Example of peers being both Origin and Branch

Metalworks Inc. sells bolts and nuts to WheelEnd LLC for use in their hub assemblies. Metalworks also buys hub castings from WheelEnd LLC, which they machine and resell. Metalworks is the origin for bolts and nuts, but the branch for hub castings. Metalworks can't just suddenly decide to be the authoritative source on hub castings, expecting WheelEnd to take changes from them; the two companies would have to agree to do this as a separate step and start over on that set of data.



👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5011&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



:

8/13/2019

Yes, it's very common for a supplier (e.g. Tenneco) to have many vendor relationships (e.g. "Made in China") but I can't think of a good reason to ever have those data pools combined.

Tenneco would probably assign new part numbers, application data, marketing copy, etc. In any case, I can't see when it would ever be published "as is".

The one exception I can think of is a "Distributor" relationship. In that case, though, they could use one (large) "secondary" data pool to collect all their supplier data, then move it to a "primary" data pool if they need to send it anywhere via Sandpiper.

So they really aren't "peers". It's always a Primary/Secondary relationship. I think maybe we could remove all the "peer", "origin" and "branch" terminology and greatly simplify the explanation of the concept.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5012&SpaceID=127)

Answer

Luke Smith (<https://autocare.communifire.com/people/autopartsource>)





8/13/2019



Doug, what do you propose would be a better pair of words besides "origin" and "branch"? In my mind, these words do pretty well do describe the positions of authoritative and non-authoritative.



Reply (/forums/post?tid=4998&ReplyPostID=5013&SpaceID=127)

Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



8/13/2019



On 8/13/2019 10:11 AM, Doug Winsby said:

So they really aren't "peers". It's always a Primary/Secondary relationship. I think maybe we could remove all the "peer", "origin" and "branch" terminology and greatly simplify the explanation of the concept.

That is a good point.

I do still want people to get the idea that this is not client-server -- for example, the primary may still ask the secondary to host the delivery method and act as what we'd traditionally call a server. They're both supposed to be equivalently *capable* pieces of software, that adapt to the requirements of the plan, and take on the role asked of them. That's sort of where I was going with the "peer" term, but I get that it has connotations that don't fit. There's probably a better word that wouldn't cause confusion.. Maybe node?

Origin and branch, I could see being loaded terms as well, but I'm a little unsure if the division between primary *actor* and primary *pool* would be confusing. I think there should be two different terms for those things but I don't have one handy. I'll make a drawing and post it with my thoughts of what the pieces would be called..



Reply (/forums/post?tid=4998&ReplyPostID=5014&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/13/2019



"the primary may still ask the secondary to host the delivery method and act as what we'd traditionally call a server"

This is the concept I'm still struggling with. My understanding of a primary/secondary relationship is that the secondary could be "promoted" to a primary, but only if the primary was off-line (at least in a traditional replication arrangement, which I know this is not).

What's wrong with the idea of having set roles with associated data pools. A company could have two different databases to act as primary or secondary if that is the requirement (as with a Distributor). Any internal movement between them could be easily accomplished (and would probably require business logic specific to the company).

The reason I'm proposing this approach is that I believe it would be much easier to implement. A simpler model means a smaller API and so less development.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5015&SpaceID=127)

Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



8/13/2019

Ah, I see. I am actually on the same page with you, for simplicity I feel and have felt that we will do better to treat the data itself as "distributed lite" -- a secondary pool is always secondary, it was part of a relationship where it came from a master, and it is assumed to be a copy. Changes made to it will be discarded once the primary source is available.

The *framework* is the flexible and independent part, not the data. The methods and the framework are distributed actors, that share a common understanding of how a fellow Sandpiper node will behave, and can make deterministic conclusions about the results of outputs. They achieve this by limiting the scope of what can be done.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5016&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/13/2019

That's how I see it. The secondary pool is a replica and so immutable. The magic is in the slices and efficient syncing.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5017&SpaceID=127)

Answer



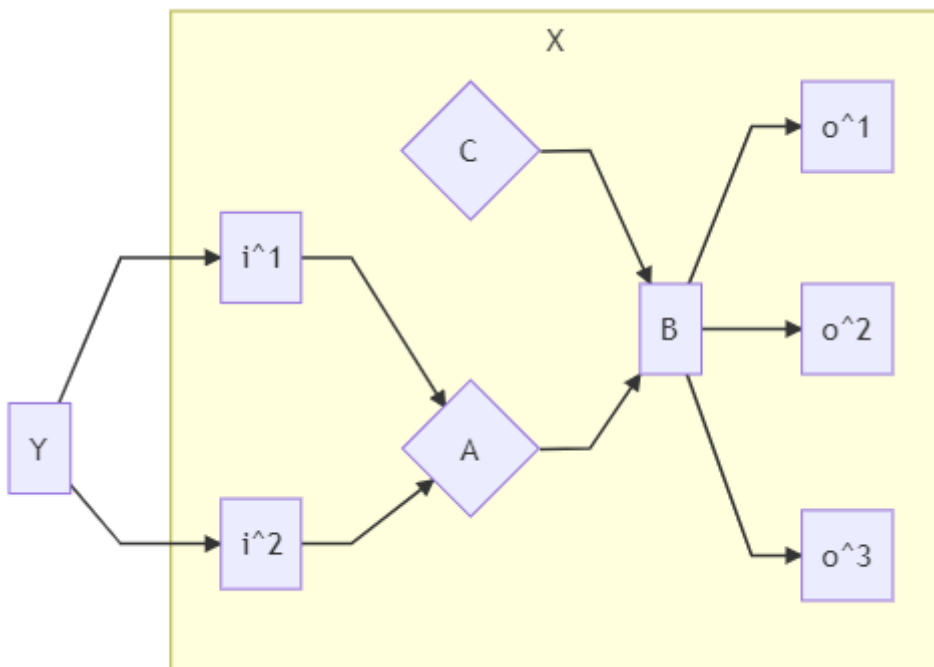
Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



:

8/13/2019

OK, here is a diagram of a Sandpiper node:



"A" is a migration process of the inputs i^1 and i^2 from other Sandpiper node Y, integrating them into the internal data, B. "C" is changes made directly to the data by users and other processes. The data in B can then be sliced and presented via Sandpiper in the outputs o^1 , o^2 , and o^3 .

Origin/Branch Way:

Node X has two secondary pools from origin Y. It integrates these into its internal databases, and performs other internal data maintenance as well. Then, as an origin with its other partners, it makes primary pools available from this internal data.

Is there a better way?

👍 Like

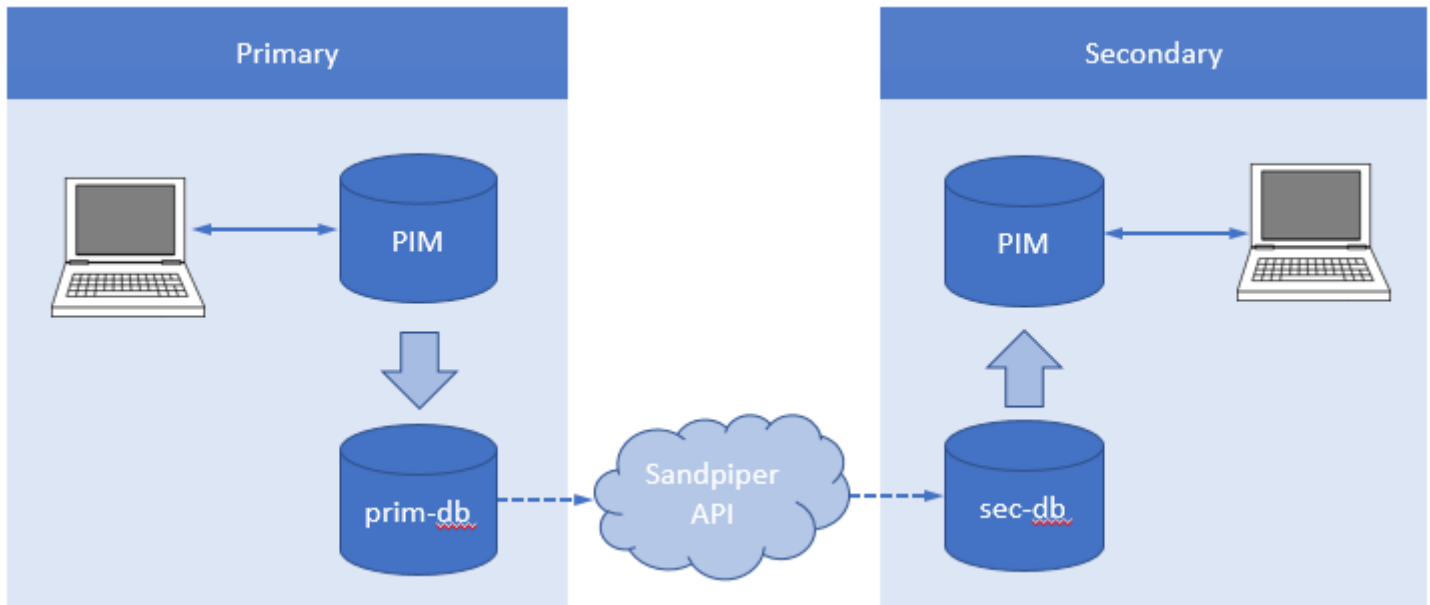


Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/13/2019

I think I understand your flow. Here's a slightly different way to look at it:



This could represent a standard supplier relationship where a supplier is "Primary" and a distributor is "Secondary". There's only one "prim-db" at the supplier and it holds everything required by its trading partners. Lots of secondary trading partners can hit the supplier's Sandpiper service to sync their "sec-db", which is then processed by their PIM (unidirectionally). This PIM update can happen whenever a sec-db change is made (i.e. near real-time), or at scheduled intervals.

When the distributor is "Primary", it moves over to the left side of the diagram. The PIM is the same, but the database used by Sandpiper is (logically) different (shown in the diagram as "prim-db").

Those vertical arrows are the "integration" steps you mentioned, and where all the (internal) value adds need to happen. The Sandpiper sync is the easy part.

👍 Like



Luke Smith (<https://autocare.communifire.com/people/autopartsource>) 

8/14/2019



Doug, your diagram is on-point. In an ideal world, the authoring PIM would *contain* SandPiper functionality, and there would be no out-bound staging prim-db. I think there's no way to avoid the sec-db staging area.



Like

[Reply \(/forums/post?tid=4998&ReplyPostID=5020&SpaceID=127\)](/forums/post?tid=4998&ReplyPostID=5020&SpaceID=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>) 

8/14/2019



Also worth noting - either side of your diagram could be the API server*. Therefore, a general-purpose software solution should implement facilities to be primary or secondary - especially a reference implementation.

**A server is a machine hosting a listening TCP socket on IP port 80 and/or 443 that is always ready to accept connections from clients and accept http "post" requests containing schema-valid SandPiper content. The server would likely be on the public internet with a static IP address and be know to public DNS (ex: sandpiper.autopartsource.com). Prudent firewall rules to limit what clients could make a connection would be advisable. A server could also be on a private (non-routable 10.x.x.x, 172.16.x.x, 192.168.0.x) network.*



Like

[Reply \(/forums/post?tid=4998&ReplyPostID=5022&SpaceID=127\)](/forums/post?tid=4998&ReplyPostID=5022&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>) 

8/14/2019



I think this will become more clear as we define the db schema and api endpoints. Maybe that's the next step.



Like

[Reply \(/forums/post?tid=4998&ReplyPostID=5023&SpaceID=127\)](/forums/post?tid=4998&ReplyPostID=5023&SpaceID=127)

Answer

Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>) 

8/14/2019





Switching gears to "**black-box payloads**", I think two of the main strengths of Luke's concept are (1) immunity to payload versioning, and (2) handling new payload standards without changing the API.

Payload versioning could be handled well with API versioning, but it still requires changes to the sync process for any payload change. My original concept required intimate knowledge of the data being synced, and I never really considered supporting non-AutoCare standards (like PartsPro and Tecdoc).

So I am on-board the black-box (or translucent-box) payload bandwagon. Still not 100% behind xml with embedded xml, but that's a small point.

👍 Like

[Reply \(/forums/post?tid=4998&ReplyPostID=5025&SpaceID=127\)](/forums/post?tid=4998&ReplyPostID=5025&SpaceID=127)

Answer

Page 1 of 2 (33 items)

[1 \(/spaces/127/sandpiper/forums/general/4998/project-requirements/1\)](/spaces/127/sandpiper/forums/general/4998/project-requirements/1)

[2 \(/spaces/127/sandpiper/forums/general/4998/project-requirements/2\)](/spaces/127/sandpiper/forums/general/4998/project-requirements/2)

[Next » \(/spaces/127/sandpiper/forums/general/4998/project-requirements/2\)](/spaces/127/sandpiper/forums/general/4998/project-requirements/2)

Copyright © 2021 Axero Solutions LLC.

Auto Care Association powered by Communifire™ Version 8.0.7789.8960



Sandpiper (/spaces/127/sandpiper) ▸ Discussions (...)

Private Space · Technology Standards (/spaces/9/technology-standards/technologyhome)

(/spaces/127/sandpiper)

Activity Stream (/spaces/127/sandpiper/?act=1)

People (/spaces/127/sandpiper/people)

Content ▼

Posted in: General (/spaces/127/sandpiper/forums/4997/general)

Project Requirements

✉ Unsubscribe from this discussion

📡 Subscribe to RSS (../..../..../..../spaces-cf/forums/rss-space-posts.ashx?

spaceID=127&topicID=4998&forumID=4997&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)



8/14/2019

The reason I'm in the xml camp is to re-use the industry-wide infrastructure we've already developed over the last 15 year. Everybody (who counts) has facilities for consuming <App id="...

Also with xml, schema validation can be inherent. The schema can be the bad-guy when someone's implementation is slightly wrong. It removes all gray-zone about standards compliance. JSON would certainly be more bandwidth efficient, but I think it would devolve into a wild-west landscape where no commercial solution is guaranteed to be fully compatible with the next. In the current file-based landscape, "*Send me your ACES file*" is backed by the full faith and credit of the schema and by proxy The AutoCare Association. If you send me something that fails xsd validation (but is almost right) I get to say: "*try again, you turkey*" while standing firmly on the moral highground.



Like

Reply (/forums/post?tid=4998&ReplyPostID=5026&SpaceID=127)

Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)



8/14/2019

I agree that the payload should be xml (or whatever the standard is, for example tab-delimited for PartsPro), but I just want us to consider using JSON for the Sandpiper wrapper.


Yes, using JSON means you need to "escape" the xml (because JSON uses quotes), but there are easy ways to do this. One brute-force method is to encode all payloads with BASE64. Or to gzip it first and then encode it. But there are other less aggressive ways to encode just the codes needing to be escaped.

We can also have our API support both methods (xml-xml or json-xml) pretty easily (using the Accept HTTP Headers <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept> (<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Accept>)) and let the receiver decide what they want to use.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5027&SpaceID=127) Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)  8/14/2019



You make some strong points - especially the tab-delimited thing. If we are willing to abide *that* nonsense, then bas64 encoding of JSON is certainly on the table as an enumerated payload type along side the good-old-fashioned ACES & PIES xml.

Watching you turn the geek-out knob to 11 with zip compression has just stripped away all my inhibitions. TOTP authentication will soon be a topic here..... :)

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5032&SpaceID=127) Answer



Doug Winsby (<https://autocare.communifire.com/people/dougwinsby>)  8/14/2019



Great, now you have me googling "TOTP authentication" <grin>.

So, a Sandpiper JSON response for a PartsPro request might look like this (if BASE64 encoded):


```
{
  "oid": "GQBFRvf112p33Q", "type": "PartsProApp", "on": "2019-08-14T09:30:10Z",
  "payload": "MQkxOTk1CTEJNQkJCQkJCQkJCQkJCQk3NDE4MglnR0MJNDU3MTA0CUQJMzQxOTQ5OzM3MDYxNzsz
MDM1MDI7MzQxNTQ5OzM1MTUwNDszNjI1NDQ7MzU0Njc1OzM1MDk4MAkxCQkKMQkxOTk1CTEJNQkJ
CQkJCQkJCQkJCQk3NDE4MglnR0MJMzM3MDA1CUQJMzQxNTQ3OzM0MTk0OTszNDE1MjE7MzUxNTA0
OzM2MjU0NDszNTQ2NzU7MzUwOTgxOzExNzYJMkQkCg=="
}
```

That payload is actually two PartsPro tab-delimited applications (an Add and a Delete). Unfortunately, BASE64 uses 33% more data than the original.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5037&SpaceID=127) Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)  8/14/2019

⋮

Yes - that is exactly what I would envision. I'm not too concerned with the bandwidth - we have good pipes these days. Also as a point of order: You didn't close your grin tag.

</grin>

You're welcome :)

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5038&SpaceID=127) Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)  8/15/2019

⋮

One other option re: actual data delivery.

For Level 1, the full files to be transferred for a slice can be available via a URI (FTP link, WWW, etc.) for the complete transfer to take place. So, the primary says, "OK, here's the place you can always find the up-to-date full set of ACES XML within this slice: <link>". Maybe with a way to do authorization credentials embedded there.

For Level 2, you could do similarly. The primary says "OK, you need new content. I have a packet containing just those IDs to add ready for you to download: <link>".

That way you never have to encapsulate.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5040&SpaceID=127)

Answer



Luke Smith (<https://autocare.communifire.com/people/autopartsource>)



:

8/15/2019

That would be fine for the "level 1 compliance", but it overly complicates level 2. You would be telling the partner system to go elsewhere for the content at the same moment that you go create the content for them to get elsewhere. Assuming we are talking about ACES & PIES XML, I think the grains in the POST input and the grains in the response output should be xsd-complaint "full" dataset that would pass validation if it were in a file. You would have the usual header and footer, etc. It would just contain a small number of apps or items.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5041&SpaceID=127)

Answer



Krister Kittelson (<https://autocare.communifire.com/people/krister-kittelson>)



:

8/16/2019

On 8/15/2019 12:38 PM, Luke Smith said:

That would be fine for the "level 1 compliance", but it overly complicates level 2.

That seems fair. I suppose it's a tradeoff between the complexity of the encapsulation vs. complexity of the operation. Encoding in BASE64 etc. is a well-known method so I could see it being preferable.

👍 Like

Reply (/forums/post?tid=4998&ReplyPostID=5043&SpaceID=127)

Answer

« Previous (/spaces/127/sandpiper/forums/general/4998/project-requirements/1)

1 (/spaces/127/sandpiper/forums/general/4998/project-requirements/1)

2 (/spaces/127/sandpiper/forums/general/4998/project-requirements/2)

Copyright © 2021 Axero Solutions LLC.

Auto Care Association powered by Communifire™ Version 8.0.7789.8960

© 2021 - AUTO CARE ASSOCIATION (<http://autocare.org>) | LEGAL & PRIVACY STATEMENT
(<https://www.autocare.org/privacy-statement/>)