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

Revised Object Model (Nee Data Model) With Flattened Structure

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/6/2020

I've spent time over the last week revising the data model to take into account Doug's suggestions about flexibility (and the concerns of too much structure) and Dave's points about making sure that a receiver can identify in the slices what pieces are necessary to make up a set that constitutes a line code.

I've attempted to unify this and the subscription object here:

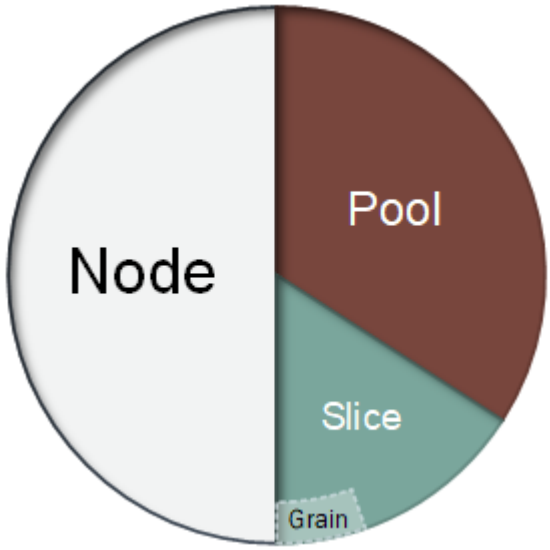
Object Model

The object model for Sandpiper defines a set of common abstractions representing the product data each Node stores. There are just three persistent objects (*Node*, *Pool*, and *Slice*), one utility object (*Grain*), and two reference objects (*Link* and *Subscription*). All Sandpiper objects have a universally unique ID that will be used for actions exclusively whenever possible.

The node represents the root of one self-contained Sandpiper environment, with one controller. It contains pools of product data, each with one owner. These pools are further subdivided into slices, each with one data scope that allows the information to be grouped and synchronized between actors.

To structure this data and aid the creation of shared scope between actors, the three persistent object types can all employ links: references to additional systems, descriptions, and data. To create the bond between actors, the secondary actor establishes subscriptions to the slices available.

Persistent Objects



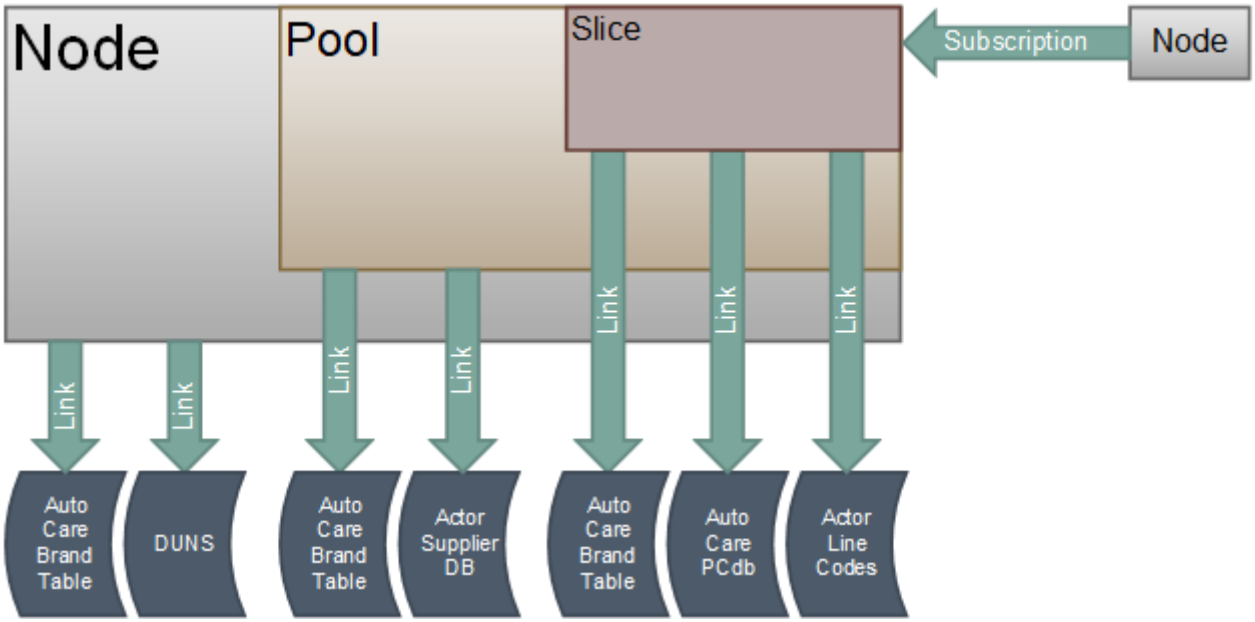
Nodes

The node is a single Sandpiper instance or system. It has a *Controller* responsible for, though not necessarily the originator of, its operation and contents.

Nodes also contain a list of part numbers, their creators, and a UUID assigned to them that will be used for all operations referencing a part number.

Pools

Within each



Sandpiper node, product data is stored in broad banks called *Pools*. These represent a business or management-level division, so that a single node might contain product data spanning multiple business approaches yet being coordinated within one system.

While a node has a controller, a pool has a *Creator*, the owner of the product data within. In some cases this will be the same as the controller, and in others it will be different. For example, if the node operator works for a shared services provider that offers data synchronization for multiple customers, the controller will be the provider, and the creator will be the customer.

Pools can be one of two types: *Canonical* or *Snapshot*.

A node's canonical pools contain the data that it owns and controls; changes made to a local node's canonical pools can be transferred to external Sandpiper nodes, with the local node as the origin point.

A node's snapshot pools contain copies of the data in other nodes' canonical pools transferred in this way. A snapshot pool is just that: a snapshot of some or all of the data in a canonical pool from an external node, at its last-known state.

Slices

A pool is divided into *Slices*. The slice is the fundamental unit of Sandpiper; basic transactions are expected to operate only on the slice, and it provides the context for all more complex transactions as well.

A slice defines the single type, format, and version of the data it contains (e.g. "Fitment", "ACES XML", "3.0"). It also defines a URI to access the data and a filename for Level 1 transactions.

Link & Utility Objects

Grains

The Grain is not a persistent object; it is created by the Sandpiper software as-needed for anchored windowing of slice data. While it is active, it belongs to only one slice. The Sandpiper system uses the *Grain Key*, a reference containing an unambiguous UUID, to safely and atomically operate on the data in pieces smaller than a whole slice.

The grain is the smallest unit on which a Sandpiper node can act.

Links

Links are references that allow slices to be tied to other systems and tagged with nonstandard metadata.

The link is the primary means of attaching overarching structure to slice data. Every partnership will have a different preferred method for establishing things like line codes, hierarchies, and sets, so the link provides a few standard methods to do this and an extensible category for what it doesn't define.

The link is also the way Sandpiper connects slice data to description or validation frameworks like reference database versions, business identities, and so on.

Subscriptions

The secondary actor in a Sandpiper relationship can subscribe to a slice, stating its intention to mirror that data and keep it synchronized with the primary actor.

This subscription includes the secondary actor's stated preference for receipt of the data: whether it should be pushed or pulled, what methods should be employed, what schedule should be followed, and what credentials will be used.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5716&SpaceID=127)

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/6/2020

To illustrate, here is a flattened table showing how an imaginary manufacturer might collect their wipers and wiper accessories. Note how the images are combined, but the applications and specifications are split, and how the links allow this to still be sortable:

Node	Pool	Pool Links	Slice	Slice Type	Slice Format	Slice Version
Auto Parts Global Gmbh	Bob's Auto Parts		Wiper Applications	Fitment	ACES Applications	3.0
			Wiper Applications	Fitment	ACES Applications	4.0
			Wiper Applications	Fitment	Flat Applications	1.0
			Wiper Details	Part Details	PIES	6.5

			Wiper Details	Part Details	PIES	6.5
			Wiper Accessory Details	Part Details	PIES	6.5
			Part Images	Digital Assets	PIES	6.5

Like

Reply (/forums/post?tid=5715&ReplyPostID=5717&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/6/2020

Here is the same thing in a pseudo-form that hints at how the elements would be present in the XML:

Node	Description
882b7d7c-a315-4488-b85d-376005fb0e5b	Auto Parts Global Gmbh

Pool	Description
6bce41d9-db09-4422-82d1-a2815f4dd5e4	Bob's Auto Parts

Slice
87446ffb-1bd0-4e19-a28d-ea995fa

		5f22bed5-fc1c-4718-aeeb-c3bf985 *693d8717-0c33-4f53-b6b7-912b76* *4223597c-d71e-48e6-98a8-9e4c3a* *1539ef2f-af4f-4543-a73c-f0fc16ce4*
		855bfd41-a18d-4e4e-82a8-ca8ba9

Like

Reply (/forums/post?tid=5715&ReplyPostID=5718&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/10/2020

> **On 2/6/2020 4:50 PM, Krister Kittelson said:**
>
> A slice defines the single type, format, and version of the data it contains (e.g. “Fitment”, “ACES XML”, “3.0”). It also defines a URI to access the data and a filename for Level 1 transactions.

In response to a concern brought up by **Dave Hastings (https://autocare.communifire.com/people/dhastings)** , I think we should relax this definition to allow multiple "type" and "formats". The slice would then become any "logical grouping" of

products, such as a PIES file and its associated digital assets.

The alternative is having some mechanism for grouping slices.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5721&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/10/2020 9:21 AM, Doug Winsby said:**
>
> In response to a concern brought up by **Dave Hastings (https://autocare.communifire.com/people/dhastings)** , I think we should relax this definition to allow multiple "type" and "formats". The slice would then become any "logical grouping" of products, such as a PIES file and its associated digital assets.
>
> The alternative is having some mechanism for grouping slices.

On a few different levels I don't like the grain being able to be different types of data within a slice.

First, practically. The slice is the point of subscription as well as the only retrievable chunk at Level 1. If one piece within it changes, the whole chunk would have to be redownloaded. This is less of an issue if it's just a few small ACES files, but what about 70GB of digital assets and a 300K ACES file in the same slice? If the ACES file changes, then the digital assets would also have to be redownloaded.

Second, conceptually. The grain is supposed to be a scalable, retargetable window *into* the data; it is not the data itself, therefore it has no file name, no file format, no size, no last change date, no author, etc.. These things all come from the slice, which *is* the data. The grain is an adaptor between the object-oriented delivery/manipulation world and the set-oriented mathematical world of the data itself. It is constantly changing and never represents an actual object.

Third, functionally. The grain should be intensely limited in its criteria. It should have one key only, and no type, and no context. If we don't keep this clear, it will (by human nature of creeping implementation scope) expand in functionality until it becomes essentially what we originally called the slice, and the slice becomes what we originally had in the section. At that point why wouldn't we just have implemented the section and been done with it?

One of the things you (rightly) brought up with the section object is that imposing too much structure in the actual model creates rigidity in an area that is extremely loose (how data creators and receivers agree to categorize data). That's where the links come in: a codified standard way to attach categorization and grouping.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5723&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

Here's a contrived example of using the links to establish categorizations. For our fictional wiper manufacturer, let's say they sell blades, arms, motors, and accessories. (For the sake of brevity I'm using a small slice ID and I've only included here the blades and the full "wipers" digital assets that covers all of these categories):

Slice ID	Description	Type	Format	Version	Filename	Primary Actor Code
1	Wiper Blades	Fitment	ACES XML	3.0	WiperBlades_Apps_ACES.xml	Wipers,Wiper Blades
2	Wiper Blades	Fitment	Flat Text	-	WiperBlades_Apps_Text.txt	Wipers,Wiper Blades
3	Wiper Blades	Part Details	PIES XML	6.7	WiperBlades_Details_PIES.txt	Wipers,Wiper Blades
4	Wiper Blades	Part Details	Flat Text	-	WiperBlades_Details_Text.txt	Wipers,Wiper Blades
5	Wipers	Digital Assets	PIES XML	6.7	Wiper_Images.zip	Wipers,Wiper Blades,Wiper Arms,Wiper Motors,Wiper Accessories

The receiver can subscribe to the pieces they want, and also use the categorization links to filter down to what they need. We can suss out the ways to help those links be more specific or less specific, but in the above, I'm assuming the codes would be multiple elements - "Wipers,Wiper Blades" would be two links, one to "Wipers" and one to "Wiper Blades". I just cut it down here to make it readable.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5724&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> **On 2/13/2020 9:09 AM, Krister Kittelson said:**
>
> The grain is supposed to be a scalable, retargetable window *into* the data; it is not the data itself, therefore it has no file name, no file format, no size, no last change date, no author, etc.. These things all come from the slice, which *is* the data.

I've been thinking of the grain completely differently. I see the grain as a syncable, black-box data container. I've currently got it defined in the database like this:

```
CREATE TABLE IF NOT EXISTS "grains" (
  "id"           uuid PRIMARY KEY,
  "slice_id"     uuid REFERENCES "slices" ON DELETE CASCADE,
  "grain_type"   grain_type_enum,
  "grain_key"    text,
  "encoding"     encoding_enum,
  "payload"      bytea,
  "source"       text,
  "created_at"   timestamp,
  CONSTRAINT "grain_type_key" UNIQUE("slice_id", "grain_type", "grain_key")
);
```

This shows that a grain belongs to just one slice. We added "grain_type" and "grain_key" mostly because of Level 1, but it also gives us the ability to add a unique key beyond the primary key. This lets us make sure we only have one grain_type for a grain_key and gives us a way to do a "replace" function. (Otherwise, the PIM would need to track all grain ids and figure out what to delete).

I just added "source" this week while working on the "sandpiper add" utility (the command line tool to add grains). I wanted the ability to see the old filename that we are replacing (as a feel-good check).

I agree that the Slice is the point of subscription and the lowest-level syncable object (all grains are synced together).

Like

Reply (/forums/post?tid=5715&ReplyPostID=5725&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

I like your wipers example. I see how it would be helpful to keep slices at that granularity (making them more reusable with different trading partners). The only change to how I have it coded now is to add a "Type" (e.g. "fitment", "part-details", etc.).

This is how I have a slice defined now:

```
CREATE TABLE IF NOT EXISTS "slices" (
  "id"            uuid PRIMARY KEY,
  "name"          text UNIQUE NOT NULL,
  "content_hash"  text,
  "content_count" integer,
  "content_date"  timestamp,
  "created_at"    timestamp,
  "updated_at"    timestamp
);
```

I already have a way to maintain meta-data for the slice (to handle reference versioning), so we could add the "Type" there or make it an actual column in the slices table.

But I'm not sure it would let us remove the "type" from the grain. I need to think that one through more.

We currently have these grain-types defined:

```
CREATE TYPE grain_type_enum AS ENUM (
  'aces-file',
  'aces-item',
  'asset-file',
  'partspro-file',
  'partspro-item',
  'pies-file',
  'pies-item',
  'pies-marketcopy',
  'pies-pricesheet'
);
```

This all comes back to the discussion on what defines a "slice". This is a key point we need to hash out. Not only what makes up a slice, but how it is maintained. That is to say, what keys we use to load it, and if Sandpiper has any responsibility there.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5726&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 11:08 AM, Doug Winsby said:**
>
> This all comes back to the discussion on what defines a "slice". This is a key point we need to hash out. Not only what makes up a slice, but how it is maintained. That is to say, what keys we use to load it, and if Sandpiper has any responsibility there.

That is what I tried to come at a bit more with the revised object model. We've been getting confused between slice and grain. In the new model, I'm trying to say something like this:

1. The slice is the sole and ultimate container for all product data
2. The grain is a utility object that gives a handle into the slice: it is a pointer to the data, not the actual home of the data
3. When a grain is communicated, it must be copied out of the slice, obviously -- and then it becomes an independent black box containing data -- but that data is brought into the slice on the receiving end and the grain is discarded

I think this gets rid of the ambiguity quite cleanly. I know it is a departure but I think our confusion over the last few months has pivoted around this and by cutting cleanly now we can avoid the gangrene later..

Like

Reply (/forums/post?tid=5715&ReplyPostID=5727&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 11:08 AM, Doug Winsby said:**
>
> Not only what makes up a slice, but how it is maintained. That is to say, what keys we use to load it, and if Sandpiper has any responsibility there.

To give my thoughts on this part rather than the model, here's what I see happening..

1. Slice has data within it, actors want to sync
2. Secondary Actor sends current change date of slices
3. Primary Actor indicates which slices need to be updated
4. For each slice:
    1. Actors agree on the level of the transaction
        1. Level 1:
            1. Secondary Actor downloads and drops all data within the slice, replacing with the new data
        2. Level 2:
            1. Primary Actor requests list of UUIDs within the slice data that Secondary holds.
            2. Primary Actor sends list of UUIDs to delete from current slice
            3. Secondary Actor deletes UUIDs
            4. Primary Actor sends grains containing copies of data to add from canonical pool slice
            5. Secondary Actor inserts grains into snapshot pool slice
5. Actors confirm transaction success

This shows a nice clear separation of the functionality; the grain becomes a tool, the slice is the permanent home.

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

I previously posted this pseudo code for a sync. We specifically wanted to stay away from **date-based** sync logic:

```
authenticate
get my.slices
for each slice in my.slices
  get slice.hash    // sha1 or md5 hash representing slice oids
  if slice.hash <> local.slice.hash    // saved from last sync or must we recalc?
    get slice.oid_list
    compare slice.oid_list with local.slice.oid_list
    for each new_oid in slice.oid_list
      get object.new_oid     // this object contains the payload
      store object in local.slice
    remove all obsolete local.slice.objects
  put slice.sync_completed   // for activity reporting purposes
```

I'm also not sure I like having different sync logic for Level 1 vs. Level 2. I think we can use Level 2 logic for Level 1 as well (it would be a complete replacement by definition, provided something changed).

Like

Reply (/forums/post?tid=5715&ReplyPostID=5729&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

That makes sense re: the sync logic. The only caveat would be supporting a non-Sandpiper-capable human downloading the full slices at Level 1 and importing them into their non-Sandpiper systems (via the Sandpiper client web UI).

So then the grain being the utility object works well: the slice itself is the whole point of a sync, and since all hashes would be run on the UUIDs contained therein, the grain doesn't enter into it except when engaged as a utility to transport or specify limited extents of product data within a slice. The grain is temporary and doesn't have to support anything but being a black box, no metadata or special considerations of file types, names, etc.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5730&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 2:01 PM, Krister Kittelson said:**
>
> on the UUIDs contained therein

Reconsidering this statement after I clicked "Post". I think for Level 1 transactions the hash would need to be full-file, since they may not have UUIDs at all.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5731&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> **On 2/13/2020 2:03 PM, Krister Kittelson said:**
>
> I think for Level 1 transactions the hash would need to be full-file, since they may not have UUIDs at all.

Every grain is assigned a UUID. Since grains are immutable, we can use the UUID to uniquely represent the payload contained in that grain. So there is no need to hash the full-file.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5732&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> **On 2/13/2020 2:01 PM, Krister Kittelson said:**
>
> The only caveat would be supporting a non-Sandpiper-capable human downloading the full slices at Level 1 and importing them into their non-Sandpiper systems (via the Sandpiper client web UI).

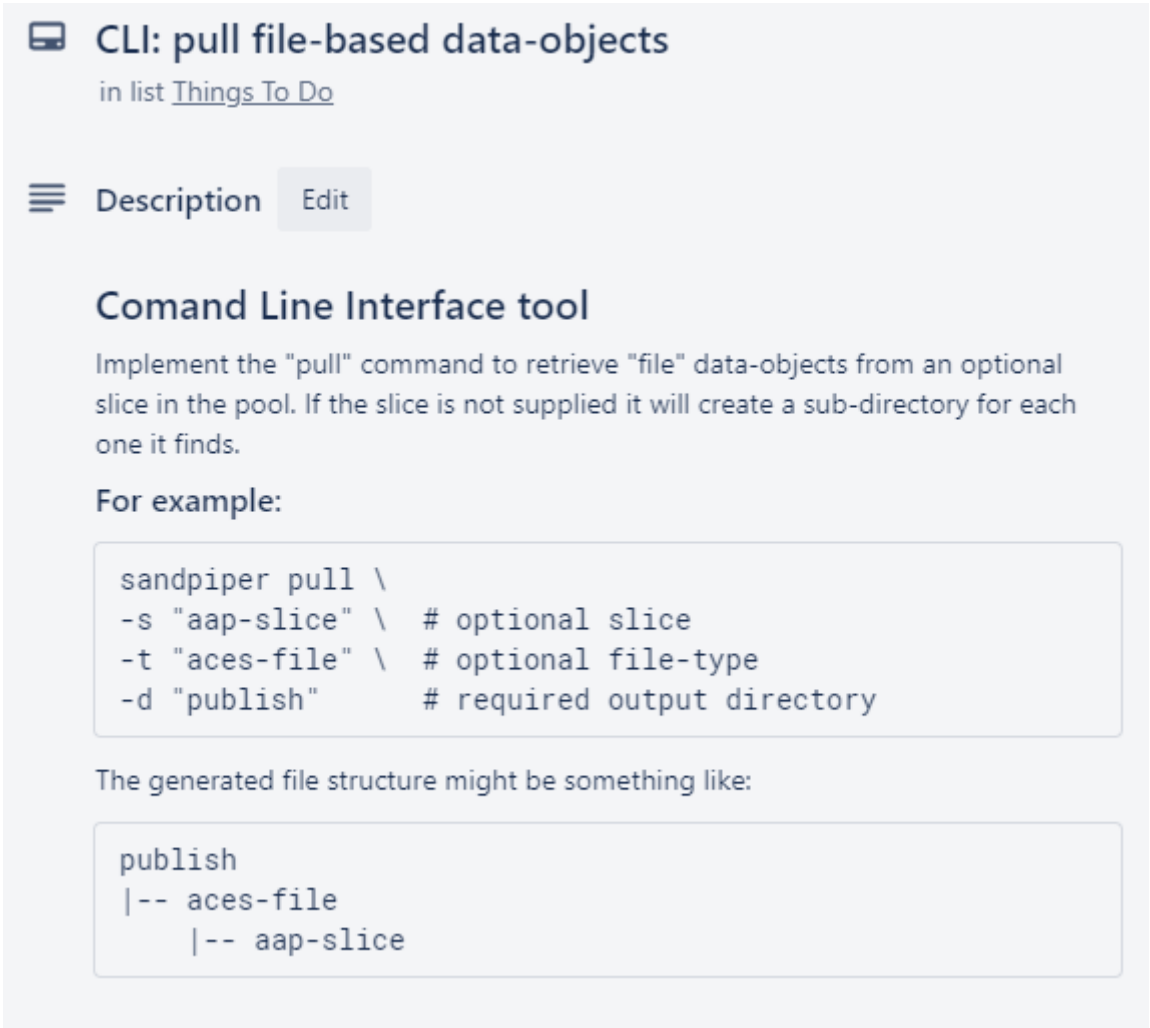
I don't see how we can enforce syncs through the web-ui (or sandpiper CLI). Not sure how the web-ui would work, but the CLI would pull **all objects** from a slice into a directory. If Level-1, it would use the new "Source" column to assign a filename.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5733&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

Here is the Trello card for Sandpiper "pull" CLI command:



Like

Reply (/forums/post?tid=5715&ReplyPostID=5734&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 2:09 PM, Doug Winsby said:**
>
> Every grain is assigned a UUID. Since grains are immutable, we can use the UUID to uniquely represent the payload contained in that grain. So there is no need to hash the full-file.

However, in the case of a Level 1 transaction, there is only one grain (the entire file). For example, an ACES file has no capacity today to do any UUIDs. So any assigned UUID would need to be updated any time the primary imports a new file, unless you're hashing and diffing it anyways. So

you're always going to be doing some kind of hash calculation on the full file for Level 1.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5735&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> **On 2/13/2020 2:22 PM, Krister Kittelson said:**
>
> So you're always going to be doing some kind of hash calculation on the full file for Level 1.

The `grain_id` has a primary key that happens to be a UUID so we know it is universally unique. When we create a grain, we assign it a new UUID. That's the one I'm referring to. Since the grain can never be changed (you must delete the old one and get a new one with a new UUID), it could take the place of a hash of the file.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5736&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 2:36 PM, Doug Winsby said:**
>
> The `grain_id` has a primary key that happens to be a UUID so we know it is universally unique. When we create a grain, we assign it a new UUID. That's the one I'm referring to. Since the grain can never be changed (you must delete the old one and get a new one with a new UUID), it could take the place of a hash of the file.

For level 1, between a primary (with a Sandpiper instance) and a secondary (without it), how would this work? Maybe:

1. Primary actor generates files out of PIM

2. Primary actor imports files into Sandpiper instance
    1. Instance drops existing slice data and imports file with new UUID
3. Sandpiper delivers files based on secondary's preferred delivery method (FTP, or waiting for the secondary to seek the files via the web interface)

Does the secondary have a way to know they've changed? How does the Sandpiper instance know?

Like you said, this needs to be put into a way that works for L2 and L1. But in the case of a L2 capability, you'll still have to scan the file somehow. Because you need to know all the UUIDs in the file.

As L1 is full-file-based, a full-file change recognition process is desirable. As L2 is UUID-based, a granular change recognition process is desirable.

Stepping outside of the example for a basic secondary and primary, for an overall process I'd propose something more like this:

1. Primary actor generates files out of PIM
2. Primary actor imports files into Sandpiper instance
    1. Instance runs a quick SHA hash of the file
        1. If it's identical to the existing file, no action.
        2. If it's different:
            1. Import the new file and update the slice's hash
            2. f the file is being imported explicitly as UUID-capable (user identifies type on import), load all UUIDs in the file and compare with existing UUIDs; otherwise create a new UUID (not as a grain, but a basic UUID list -- that's all we need)
            3. Perform update and delete inside instance however is best (which could include grains used as a window in a mini-L2 transaction like)
3. Sandpiper delivers based on secondary preference (including L2 sync if set up)

Reply (/forums/post?tid=5715&ReplyPostID=5737&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

A few things.

1. Sandpiper doesn't support ftp delivery (at least not that I'm aware of).

2. For Machine-Human sync (Level 1 only), it is just a delivery method and so must deal with filenames. That is why we store the filename in the "Source" column of the grain. When the file is extracted (by the user-interface), it will save the grain as a file with the source filename. (What's the alternative, we can't save a native grain to the file system).

3. For Machine-Machine sync (including Level 1), the grain is moved from one database to the other (without any change). They could then extract from their own data pool using the CLI tools as explained above in Machine-Human.

4. The grain is a black box. We don't know if it contains UUIDs or anything else. The only UUID is on the primary key of the grain. We should never need to hash file contents.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5738&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

> **On 2/13/2020 3:13 PM, Doug Winsby said:**
>
> 4. The grain is a black box. We don't know if it contains UUIDs or anything else. The only UUID is on the primary key of the grain. We should never need to hash file contents.

I have always thought that the grain is a black box for data with the exception of its key, which is a reference to a unique UUID contained in the data.

If the grain is being divided by part number, then the grain key is a reference to a unique ID tied to that part number -- the reason for the part list with UUIDs. At no time should the grain *itself* care about the data, but it should only contain or reference data with a primary key

UUID that matches the grain key value.

Therefore the grain has to contain values that are part of the data, and if Sandpiper is ever to operate at Level 2 it needs to be able to understand how to get to a UUID inside the data, format by format. And to do that, it either needs to be given the data directly from the PIM, or parse a file, or both.

This seems to me to restate the importance of removing the grain from being a persistent object. It's blurring the line between the slice and itself, and operating kind of like a slice and kind of like a subdivision of a slice. And it's enabling subdivision of data even though Sandpiper can't ever subdivide the data, because it has no way to parse the data.

To handle the sync without grain persistence in L2 is a different method than with grain persistence. I recognize that this means a lot of rework. I think it is going to pay off in future maintainability and avoided spaghetti however.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5739&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/13/2020

My thought on synchronizing using the new object model. Pseudocode:

```
for slice in secondary.slices:
        primarySlice = primary.slices[slice.uuid]

        if slice.hash != primarySlice.hash      # We have an update to make

                grains = slice.subdivide_by(slice.subscriptions.grainType)
                primaryGrains = primarySlice.subdivide_by(grains.grainType)

                # Process deletes
                for grain in grains not in primaryGrains:
                        grains.delete(grain)

                # Process additions
                for grain in primaryGrains not in grains:
                        grains.add(grain)
```

For L1, the subscription grainType would be "Full" and return a single grain object, that windows the entire file. The grain itself doesn't exist except when called upon, and can be sliced according to the methods available to the subscription.

(I'm also assuming using set-based for loops, which depends on the language features)

Like

Reply (/forums/post?tid=5715&ReplyPostID=5740&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> "I have always thought that the grain is a black box for data with the exception of its key, which is a reference to a unique UUID contained in the data."

I'm not sure how it helps us to assume anything about the payload. They are just blobs that Sandpiper moves around.

> "If the grain is being divided by part number, then the grain key is a reference to a unique ID tied to that part number -- the reason for the part list with UUIDs. At no time should the grain *itself* care about the data, but it should only contain or reference data with a primary key UUID that matches the grain key value."

I think "divided by part number" refers to grain_types that end in "_item", such as "aces_item", right? If that is the case, then I agree with this statement. The "grain_key" for a "pies_item", for example, would be something that identifies the "item" to the PIM. This could be a primary part-number, or maybe a UUID tracked by the PIM. But it really has no meaning to the receiver.

I agree the grain is not "self-aware". But the contents of the grain should have meaning to systems that both create the grain and those that consume the grain.

> "... if Sandpiper is ever to operate at Level 2 it needs to be able to understand how to get to a UUID inside the data, format by format."

I disagree.

> "... restate the importance of removing the grain from being a persistent object."

How can the grain not be persistent? It is what is delivered.

> "To handle the sync without grain persistence in L2 is a different method than with grain persistence."

I don't know what that means.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5741&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/13/2020

> "For L1, the subscription grainType would be "Full" and return a single grain object, that windows the entire file. The grain itself doesn't exist except when called upon, and can be sliced according to the methods available to the subscription."

You've completely lost me there. Are you saying that the data to sync is dynamically created somehow from a master set of data?

Like

Reply (/forums/post?tid=5715&ReplyPostID=5742&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/14/2020

> **On 2/13/2020 9:09 AM, Krister Kittelson said:**
>
> First, practically. The slice is the point of subscription as well as the only retrievable chunk at Level 1. If one piece within it changes, the whole chunk would have to be redownloaded. This is less of an issue if it's just a few small ACES files, but what about 70GB of digital assets and a 300K ACES file in the same slice? If the ACES file changes, then the digital assets would also have to be redownloaded.

It is not true that all grains in a slice must be downloaded if only one changes. Actually, there is no concept of a "change". There is only existence. If a grain exists in the primary and the secondary doesn't have it, then the secondary will ask for it. That is how the sync works.

The only reason we have hashes at all is to avoid asking for a list of grain IDs. We don't actually need the hashes, the sync would work fine without it.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5743&SpaceID=127) Answer

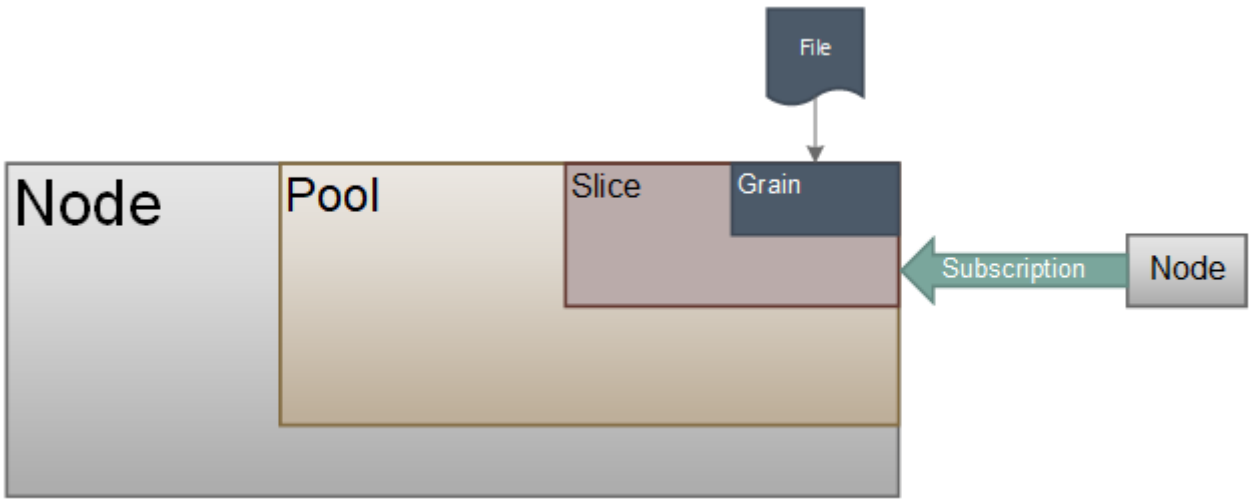
Posted in: General (/spaces/127/sandpiper/forums/4997/general)

Revised Object Model (Nee Data Model) With Flattened Structure

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/14/2020

> **On 2/13/2020 4:24 PM, Doug Winsby said:**
>
> You've completely lost me there. Are you saying that the data to sync is dynamically created somehow from a master set of data?

If I understand correctly, the model you're using looks like this:



In this model, every grain is a file, and every slice is just a group.

First, how do you do synchronization at the ACES item or individual part level, like in the grain types you defined? For my ACES output in Raybestos, I have just shy of 1M applications. To be able to send these as individual application changes, would the PIM have to create 1M individual files, and then load them into sandpiper individually?

It seems like you're saying that Level 2 is sort of a slightly-transport-driven Level 1; it's still complete file updates. It's just that we chop the files down really really small. Is that sustainable and scalable? I have several brands and am adding more every day, growing now towards 4M applications overall not counting NAPA.

Level 2 is supposed to be able to operate on the data at a granular level without understanding its structure except in the barest form: a single, primary UUID, and its associated (blind) content. But Sandpiper will still need to be able to intelligently chop the data up into pieces with a key value pulled from a single UUID.

It's not that we know all the elements inside an ACES <App> element. It's that you know that the <App> is the boundary to chop the file, and the key to extract is the (currently hypothetical) uuid attribute. All the content goes into a database as a row with a blob field and a key field.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5744&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/14/2020

> **On 2/14/2020 9:47 AM, Krister Kittelson said:**
>
> It's not that we know all the elements inside an ACES <App> element. It's that you know that the <App> is the boundary to chop the file, and the key to extract is the (currently hypothetical) uuid attribute. All the content goes into a database as a row with a blob field and a key field.

Looking at this, I could see the grain persisting in this way. Maybe we define the grain type per slice, and then the grain can exist but with a known size and shape?

Still, the grain as a file is not the right approach because the grain is supposed to be the smaller, operable update -- and when we get into millions of records this doesn't work well as files.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5745&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/14/2020

As I understand it, L2 supports "item-level" syncing (not application-level). The reason for "item-level" is that it allows a logical grouping around a natural data point (a sellable unit). You want to make sure you have everything needed to sell that part.

The burden is on the PIM to keep these grains updated.

For L1, it is designed to be a painless transition from what they're doing now. It is file-based. The entire file is zipped (encoded) into a grain's payload.

> "Sandpiper will still need to be able to intelligently chop the data up into pieces with a key value pulled from a single UUID."

I feel this is the PIM's job. Sandpiper is just a delivery mechanism.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5746&SpaceID=127) Answer

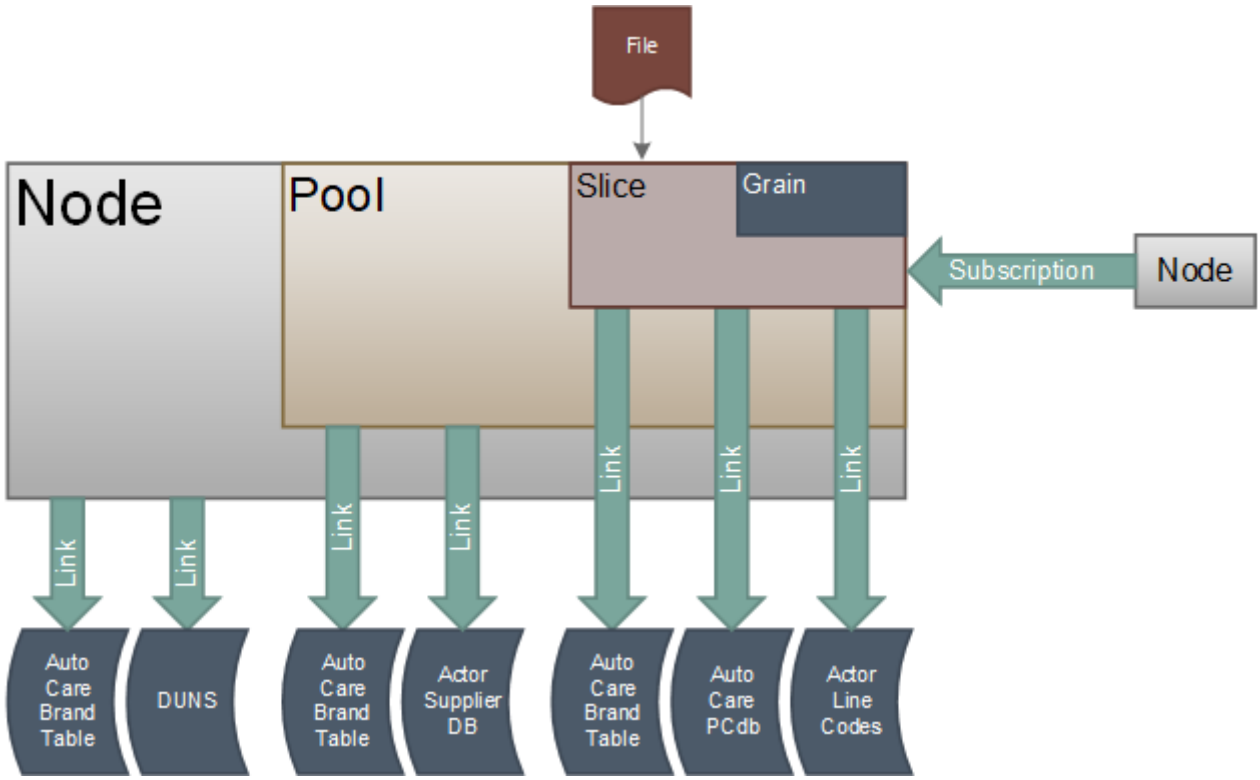
Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
2/14/2020

> **On 2/14/2020 10:09 AM, Doug Winsby said:**
>
> I feel this is the PIM's job. Sandpiper is just a delivery mechanism.

I think it's unrealistic, however, to expect a PIM to create thousands of physical files to do this separation. I am not saying that we have to have Level 2 ready to go and we need to have a full DB solution at hand, but this kind of segmentation is going to be problematic.

I want us to stay away from the grain as a file; that's not a grain, it's just shifted the slice up a level to become essentially what the Section was in the old data model. This is not what the grain is, conceptually; a grain is a small piece, an indivisible chunk of data. The file is the grouping of that data -- it's the slice.

I am open to changing the concept and returning the grain to be persistent, however; to treat it as a proper object, with its division statically defined by the slice.



Like

Reply (/forums/post?tid=5715&ReplyPostID=5747&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
2/14/2020

> "I think it's unrealistic, however, to expect a PIM to create thousands of physical files to do this separation."

Sandpiper provides an API for creating subscriptions, slices and grains. The PIM uses these API resources to create syncable objects that are delivered to trading partners.

There are no physical files in Sandpiper, only grains that have a payload. It is not unrealistic for a PIM to create these grains, no matter how many there are. The beauty is that the PIM can create them and then not worry about delivery.

Whether or not a slice allows more than one file-based grain is a small point (at least from an implementation standpoint). I personally think it makes sense to allow the slice to group L1 file-based grains, but I'm not adamant on that point.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5748&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/9/2020

Answered

After much (online & offline) debate, it was decided that we would make **all grains in a slice have the same "content-type"**.

The reasoning is that it would allow **slice reuse** across subscriptions, and so you could "build up" a subscription from more targeted pre-defined slices. The down-side is that you must now use some other way to "group" related slices so the receiver can consume them as a logical chunk.

So, instead of a slice being something like "Wiper Products", it would be "Wiper Blades" containing only "aces-items". You would define another slice for "Wiper Blades" containing "pies-items". We would then use slice metadata (in addition to a subscription) to process them as a logical group of some sort (e.g. ACES and related PIES data).

One thing I don't like about this change is that our **slice names now become clunky** (because they must be unique). They should probably include the content type in the name. So "Wiper Blades" would become "Wiper Blade Applications (ACES)", "Wiper Blade Applications (NAPA)" and "Wiper Blades Product Data (PIES)". Of course, in reality, we might need to include customer specific information. So you might have "Wiper Blade Applications (AutoZone)" which implies ACES, but has AZ part numbers, for example.

Here is the list of slice "content-types" that I'm considering:

```
CREATE TYPE content_type_enum AS ENUM (
  'aces-file',
  'aces-items',
  'asset-archive'
  'asset-files',
  'partspro-file',
  'partspro-items',
  'pies-file',
  'pies-items',
  'pies-marketcopy',
  'pies-pricesheets'
);
```

Notice that some of them are plural (e.g "aces-items") to imply you can have more than one grain of that type in a slice (needed for Level 2). The Level 1 (file-based) ones are singular to imply you can only have one grain for those.

I added "asset-archive" to handle L1 assets delivered as a zip file, but I'm concerned about the size of that one. I'm going to do some postgresq research before finalizing it. I also have "asset-files" for L2 delivery of individual jpeg files, for example. (I don't like that it ends in "files" because it looks a bit like an L1 file-based type, but I couldn't come up with a better name ("asset-blob" just seemed confusing)).

I've started making these changes to the reference implementation code.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5804&SpaceID=127) Not answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/23/2020

One problem I'm seeing while implementing this change (moving content-type from the grain to the slice) is that there's now no way to enforce that all grains in a slice are the same type. This means you could accidentally (or on purpose) have an **aces-based grain** in a **pies-based slice**.

I think maybe it would be a good idea to keep grain-type on the grain **as well as** adding the new content-type to the slice.

I guess it really wouldn't keep someone from adding pies content to an aces-type grain, so maybe the question is moot. (But I just feel the data container itself should know what is in it).

Thoughts?

data-consistency (/spaces/127/sandpiper/searchresults?keyword=%22data-consistency%22&searchtags=1)
grain (/spaces/127/sandpiper/searchresults?keyword=%22grain%22&searchtags=1)
grain-type (/spaces/127/sandpiper/searchresults?keyword=%22grain-type%22&searchtags=1)

Like

Reply (/forums/post?tid=5715&ReplyPostID=5837&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/29/2020

Another problem with the **one grain-type per slice** model is the "pies-pricesheet" grain. We might want to reconsider how we deliver this segment of PIES.

It feels wrong to require a slice for something that might change often, requiring a new slice subscription anytime a pricesheet changes.

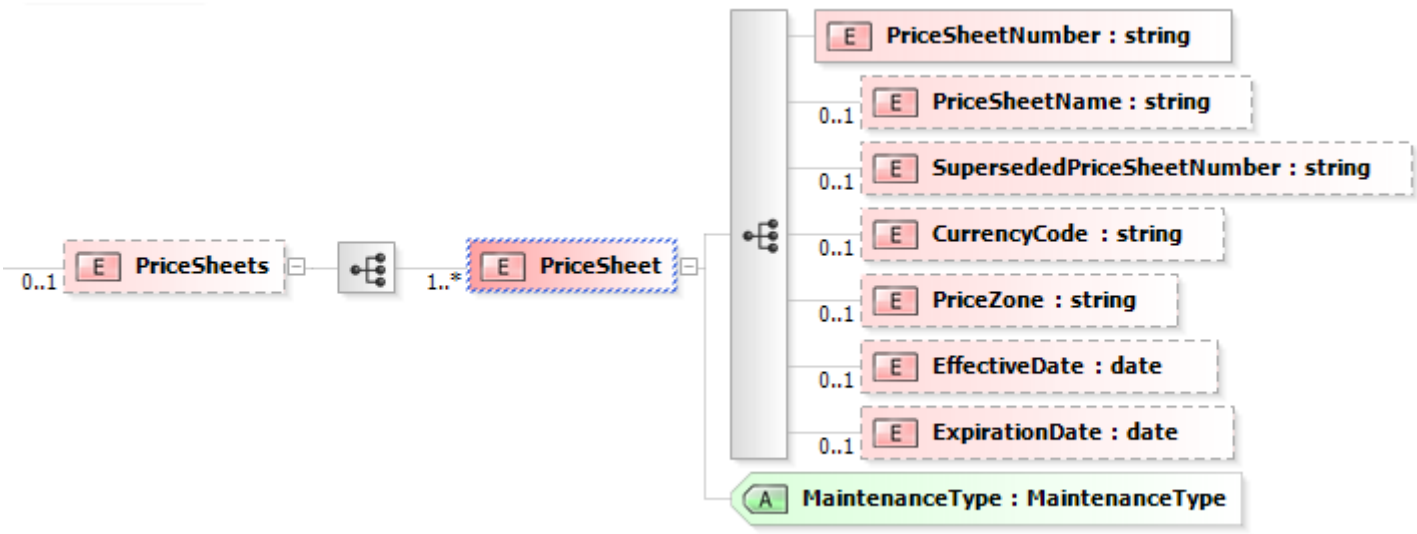
Maybe we should handle it as a "reference version" like we do for VCdbVersionDate. This would mean that we move it to the "slice metadata" which can change without creating a new slice.

The problem is that the pricesheet segment is fairly complex:

A50	PRCS	Price Sheet Segment	O	O		<PriceSheet>	
A51	PRCS	Maintenance Type	M	R	ID1	MaintenanceType	A
A52	PRCS	Price Sheet Number	KM	R	AN1/15	<PriceSheetNumber>	2009WD
A53	PRCS	Price Sheet Name	O	O	AN1/30	<PriceSheetName>	January 2009
A55	PRCS	Superseded Price Sheet Number	O	O	AN1/15	<SupersededPriceSheetNumber>	2008WD
A60	PRCS	Currency Code	O	O	ID3	<CurrencyCode>	USD
A65	PRCS	Price Zone	O	O	A 1/10	<PriceZone>	Western
A70	PRCS	Price Sheet Level Effective Date	O	O	D	<EffectiveDate>	2009-01-03
A75	PRCS	Price Sheet Level Expiration Date	O	O	D	<ExpirationDate>	2009-01-03

My first thought is to use JSON to encode this information and store it in a "pies-pricesheet" metadata key. (The reason for JSON instead of XML is because the metadata is allready JSON and so it would be easier for the API client to parse).

The `PriceSheets` segment actually looks like this:



So we would need to allow multiple "PriceSheet" elements. Maybe something like this:

```
[
  { "PriceSheetNumber": "WD2009-A", "PriceZone": "A" },
  { "PriceSheetNumber": "WD2009-B", "PriceZone": "B" }
]
```

Thoughts?

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

Like

Reply (/forums/post?tid=5715&ReplyPostID=5852&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
3/30/2020

> **On 3/23/2020 4:20 PM, Doug Winsby said:**
>
> One problem I'm seeing while implementing this change (moving content-type from the grain to the slice) is that there's now no way to enforce that all grains in a slice are the same type. This means you could accidentally (or on purpose) have an **aces-based grain** in a **pies-based slice**.

I think having the content type on the slice only actually makes sense, because having the grain type on the grain only duplicates the data. Someone could, as you noted, actually just throw data in the grain that's not appropriate anyways -- even with the grain type set to ACES file. Having the field doesn't help enforce this any more...

In terms of the implementation schema, it's likely harmless to have the field remain duplicated, but it may eventually create issues where someone misconstrues the presence of the grain type to mean that they should actually be allowed to put any type in regardless of the slice grain type, and then only run into issues when they try to parse the data externally.

So I'd say it may be better to remove it and pursue some other means of validating the data if necessary. Like you've rightly noted elsewhere, understanding the data inside the slice is kind of the responsibility of the PIM or pre-processor rather than the sandpiper server itself.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5854&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
3/30/2020

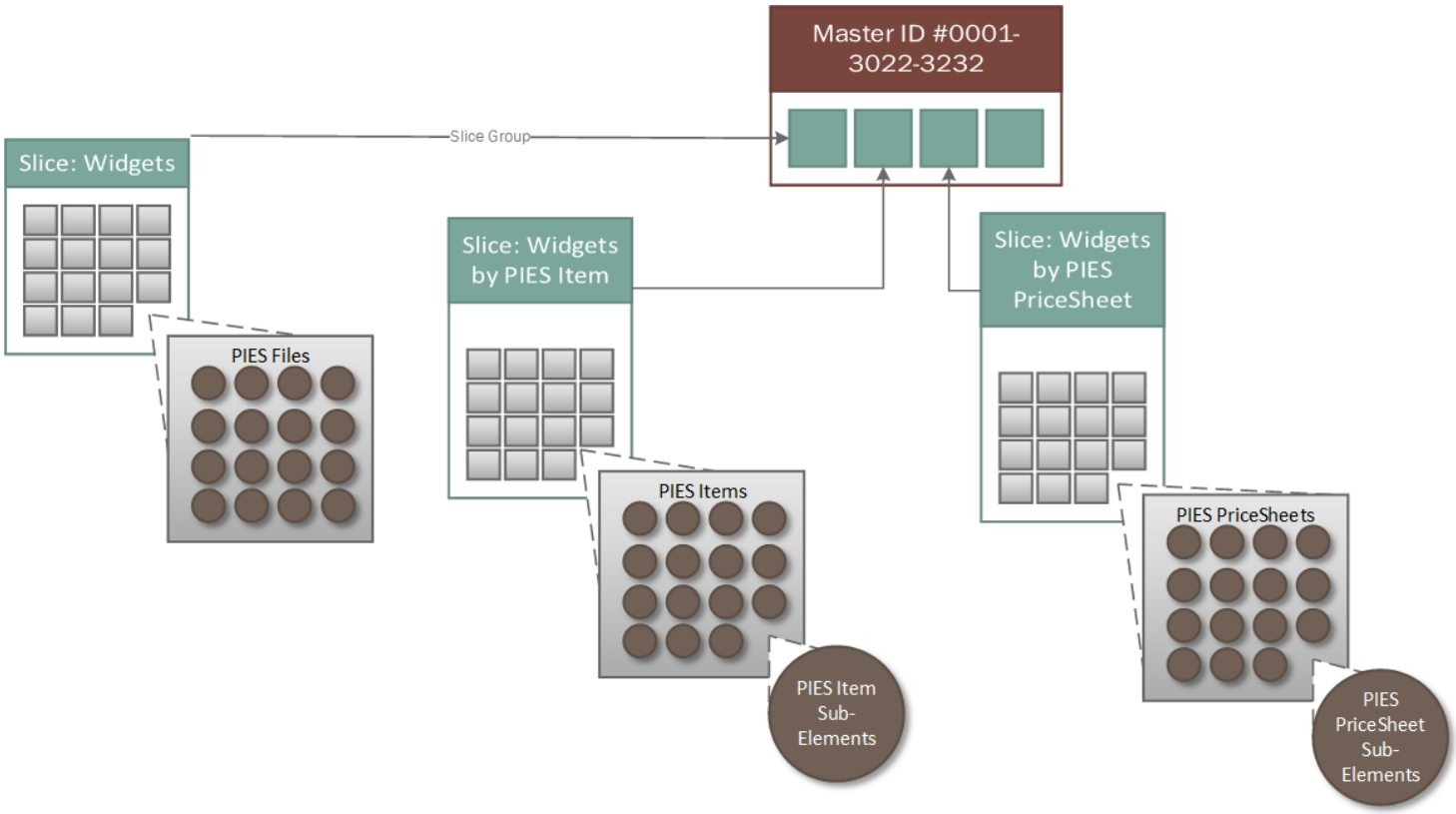
> **On 3/29/2020 10:03 AM, Doug Winsby said:**
>
> Another problem with the **one grain-type per slice** model is the "pies-pricesheet" grain. We might want to reconsider how we deliver this segment of PIES.
>
> It feels wrong to require a slice for something that might change often, requiring a new slice subscription anytime a pricesheet changes.

A good point about PIES. We've so far focused on these files in their total contents but not the pieces. PIES presents a unique challenge because it has these segments.. that won't neatly granulate.

For Level 1 we're OK because these are full files. But for Level 2 transactions, that's where this becomes more meaningful. How do we granulate a file in multiple ways yet preserve the mathematical set, which requires all elements to be equivalent?

I agree with you that the metadata / link seems to be the place to make this connection. Though I think I'd take a different direction slightly.

This is making me think that there's actually a granulation strategy here. If we want to split a file into pieces, for a PIES file you have two options: by PIES segment (Price Sheet, Market Copy, Items), or by a specific granulation of one of those segments (PIES Items being split into Item grains, PIES price sheets being split into PriceSheet grains, etc.). The secondary node could subscribe to multiple slices of the same PIES file.



So a node subscribes to whatever pieces they like, and knows they're getting the data that's a subset of the full file via the links (metadata) as you mentioned.

We would need to come up with a name for this link type though. Something like "MasterSet".

Like

Reply (/forums/post?tid=5715&ReplyPostID=5855&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/30/2020

I believe we settled on **part-number granularity** for all L2 grains. This seemed to be the "just right" level taking into account reference-versions and the need for a receiver to process data by a complete set based on a well understood grouping (the Item).

Your thought about "PIES price sheets being split into PriceSheet grains" works as long as you allow all of them to be contained in a single slice (which is an interesting idea). I was thinking we must allow only one per slice, which is untenable. But **if we allow multiple, it works quite nicely**. The grain-key would just be the PriceSheetNumber. When a PriceSheet is obsolete, the grain is removed. Nice!

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

Like

Reply (/forums/post?tid=5715&ReplyPostID=5859&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/30/2020

I think you're right about not duplicating the grain-type (or more accurately now called slice-type or content-type). I'll make that change.

I'm also going to change the `sandpiper add` command to remove the `--grain-type` parameter. So it would now look like this:

```
sandpiper add \
-slice "aap-brake-pads"  \ # slice-name
-key "brakes"            \ # grain-key
-noprompt                \ # don't prompt before over-writing
acme_brakes_full_2019-12-12.xml  # filename to add</pre
```

So the grain is always defined by the slice it's added to.

remove-grain-type (/spaces/127/sandpiper/searchresults?keyword=%22remove-grain-type%22&searchtags=1)
sandpiper-add (/spaces/127/sandpiper/searchresults?keyword=%22sandpiper-add%22&searchtags=1)

Like

Reply (/forums/post?tid=5715&ReplyPostID=5860&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
3/30/2020

> **On 3/30/2020 10:55 AM, Doug Winsby said:**
>
> I believe we settled on **part-number granularity** for all L2 grains. This seemed to be the "just right" level taking into account reference-versions and the need for a receiver to process data by a complete set based on a well understood grouping (the Item).

We did, though as a step towards Level 2.. But with our updated models it seems like we have the ability to do this now, using the grain key, as long as we keep the boundaries well-defined. The grain key is a supposed to be a UUID (we haven't hashed that out so tightly but the idea is that part numbers etc. are all just text, and so the node will contain a list of the part numbers -- if we expand that to be a list of the grain keys instead, it works the same).

Like

Reply (/forums/post?tid=5715&ReplyPostID=5861&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
3/30/2020

> **On 3/30/2020 11:45 AM, Doug Winsby said:**
>
> I think you're right about not duplicating the grain-type (or more accurately now called slice-type or content-type). I'll make that change.
>
> I'm also going to change the `sandpiper add` command to remove the `--grain-type` parameter. So it would now look like this:
>
> ```
> sandpiper add \
> -slice "aap-brake-pads"  \ # slice-name
> -key "brakes"            \ # grain-key
> -noprompt                \ # don't prompt before over-writing
> acme_brakes_full_2019-12-12.xml  # filename to add</pre
> ```
>
> So the grain is always defined by the slice it's added to.

Seems good to me!

Like

Reply (/forums/post?tid=5715&ReplyPostID=5862&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
3/30/2020

> **On 3/30/2020 11:56 AM, Krister Kittelson said:**
>
> The grain key is a supposed to be a UUID (we haven't hashed that out so tightly but the idea is that part numbers etc. are all just text, and so the node will contain a list of the part numbers -- if we expand that to be a list of the grain keys instead, it works the same).

The grain-key and grain-id are two different things. The grain-id is a uuid which has no meaning (other than a way to uniquely identify that particular grain). The grain-key is a text field, defined by the creator, to help the PIM make updates (deleting old grains).

The grain-key may not have an L1 use now that you can only have one file per slice. I'll need to think that through more.

> "the node will contain a list of the part numbers"

Were you thinking that was stored as an actual structure somewhere? I don't have any way of tracking this currently.

grain-id (/spaces/127/sandpiper/searchresults?keyword=%22grain-id%22&searchtags=1)
grain-key (/spaces/127/sandpiper/searchresults?keyword=%22grain-key%22&searchtags=1)

Like

Reply (/forums/post?tid=5715&ReplyPostID=5863&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
4/1/2020

> **On 3/30/2020 12:19 PM, Doug Winsby said:**
>
> Were you thinking that was stored as an actual structure somewhere? I don't have any way of tracking this currently.

You know, I was, but now that I'm thinking about it I wonder if UUID storage for keys is just a duplicate structure anyways...

Originally we were thinking that part numbers are just text, and text is prone to interpretation/encoding/etc. issues. But those same issues would exist if the node places them in a list of UUID to part number.. and it adds a lot of complexity that maybe is not necessary.

Perhaps I'll modify to suggest that UUID is preferred but at this time the grain key is free text (which can accommodate UUIDs anyways). I'd like someday to figure out a way to do this but I don't know that I have any better ideas, and it complicates the plan.

Like

Reply (/forums/post?tid=5715&ReplyPostID=5869&SpaceID=127) Answer