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

Level 2 Content Granulation Pattern

Unsubscribe from this discussion
Subscribe to RSS (../../../../../../spaces-cf/forums/rss-space-posts.ashx?spaceID=127&topicID=6020&forumID=4997&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
6/22/2020

Hello, All,

As promised on Friday, here's the revised information that you helped me sort out mentally.

I've abandoned the idea of the index slice (where the slice's content is purely a pointer to the full slice) for L2 because this gets really complicated and I'm not sure what I was thinking.

Instead I've shifted to what I've called the "Master-Derived Group" pattern. Here's the writeup I'd like to add to the implementation section of the documentation:

Granulation: the Key to Level 2

A Level 1 slice contains a full file that can only be communicated in full. But what if we want to break it into parts that can be synchronized individually? This becomes possible in Level 2 transactions, where we can more finely subdivide the data in a full file and use these pieces to deterministically synchronize primary and secondary data pools.

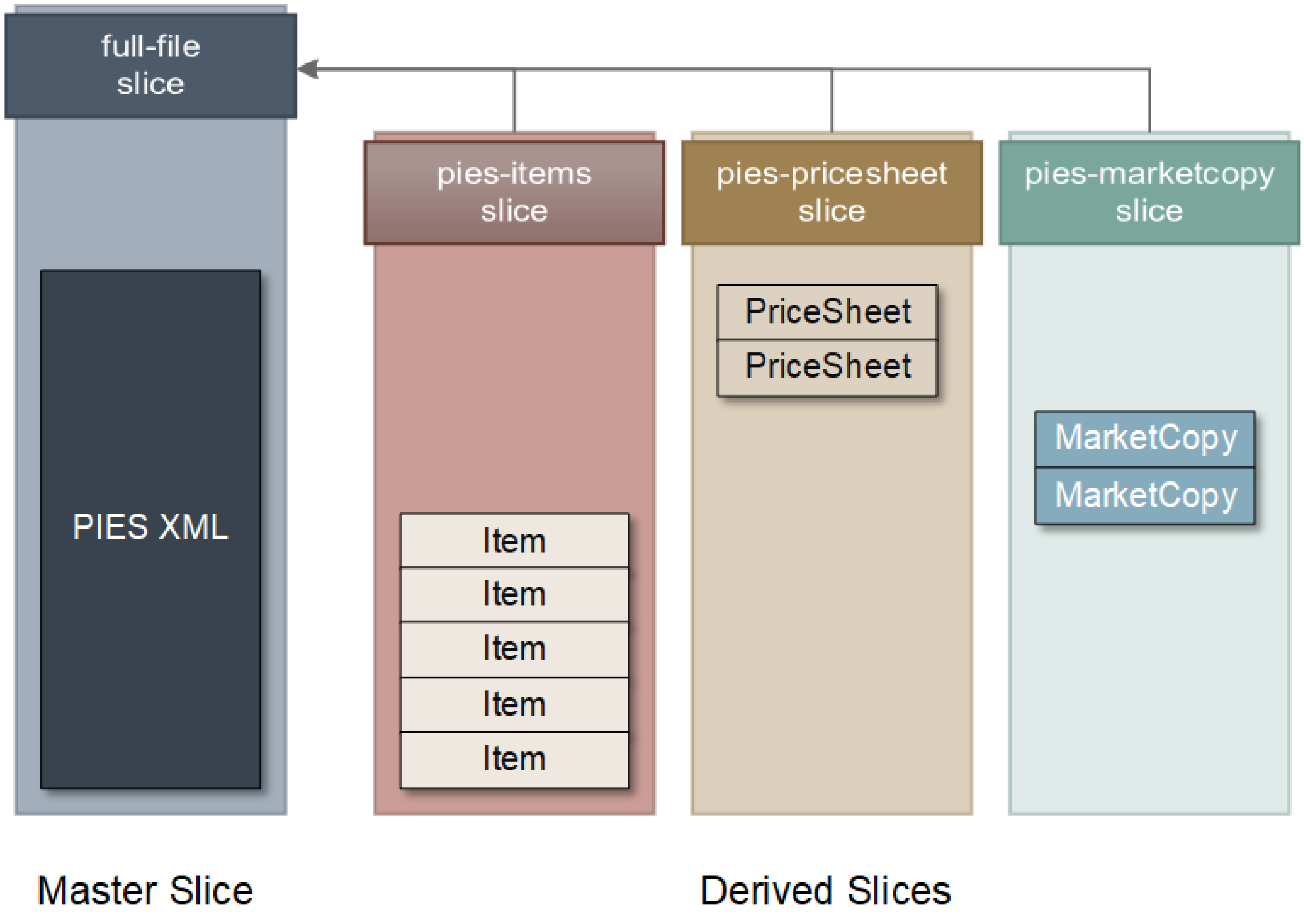
To do this cleanly, we'll define a master slice and one ore more derived slices.

The Master-Derived Group

The *Master-Derived Slice Group* allows a Sandpiper node to support both Level 1 full file slices and Level 2+ operations on that same data. Once we enter Level 2, we begin to work more directly with grains.

All slices technically contain at least one grain; in the case of a Level 1 file, it's just that there's one single, large grain: the full file. But for Level 2 slices, the information in the full file can be broken into grains that allow different perspectives of access and synchronization, e.g. by PIES item, by ACES item, and more.

In the master-derived group pattern, we define a single Level 1 *Master Slice* with the full dataset, and one or more *Derived Slices* containing granulated data extracted from the master. The derived slices reference the master's unique ID using a link with system "Master", indicating their source. These derived slices use the *Slice Type* to indicate the kind of division they do (i.e. what kinds of grains are within), and actors can subscribe to them directly.



For example, in a PIES file, we might want to work with PIES items (which will use the part number as a key) as well as PIES market copy and PIES price sheets (which do not reference part numbers and have to be keyed differently). In this case we'd have four slices: the full master, a derived slice by items, a derived slice by price sheets, and a derived slice by market copy.

In this way, a file with multiple segments can be broken out for processing at Level 2, yet Level 1 remains accessible and useful for full set analysis (or file areas that cannot be easily granulated, like prerolls and headers).

Like

Reply (/forums/post?tid=6020&ReplyPostID=6021&SpaceID=127)

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
6/22/2020

A few thoughts:

1. We should change "master" to "parent" (or other relationship terminology).
2. It might help to show an example of the slice-metadata for each of these slices. Maybe in the next section so as not to confuse this (very clear) explanation.
3. I'm not sure I really understand the need to link smaller-grained ("derived") slices to their full-file parents. I wouldn't expect a receiver that is taking L2 from a company to also take L1 for the same information. Shouldn't meta-data on the L2 slice be all that is required by a receiver? Can you give another example why a receiver would need this link?

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

1 Like

Reply (/forums/post?tid=6020&ReplyPostID=6022&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
6/22/2020

This discussion has made me think a bit more about "slice links". I see the value in having a way to show related slices, but I wonder if we are building a possible dependency into the design. Once you define the relationship, how do you know they are both kept in sync with each other?

I was thinking it might be nice, for example, to link image slices to a PIES digital-asset slice (pretending there was such a thing). But I think that would be a mistake. It would just be building in more indirection and unnecessary dependencies. The standards should do the linking, I think.

Like

Reply (/forums/post?tid=6020&ReplyPostID=6023&SpaceID=127)   Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
6/22/2020

> **On 6/22/2020 11:51 AM, Doug Winsby said:**
>
> We should change "master" to "parent" (or other relationship terminology).

I thought some about this too, how about we change to "Root Slice" and "Derivative Slice"?

> **On 6/22/2020 11:51 AM, Doug Winsby said:**
>
> It might help to show an example of the slice-metadata for each of these slices. Maybe in the next section so as not to confuse this (very clear) explanation.

I think this makes sense. I'll write it up..

> **On 6/22/2020 11:51 AM, Doug Winsby said:**
>
> I'm not sure I really understand the need to link smaller-grained ("derived") slices to their full-file parents. I wouldn't expect a receiver that is taking L2 from a company to also take L1 for the same information. Shouldn't meta-data on the L2 slice be all that is required by a receiver? Can you give another example why a receiver would need this link?

In addition to providing a convenient handle for sandpiper nodes that are upgrading from Level 1 to Level 2, I think the link has two needs it serves:

1. External granulator hook (this also comes in to the other reply you posted)
    1. A granulator can automatically process and update slices using the full file. Example, if I have a full ACES file and I have a granulator that will also chop that into app items, I can update the full file and then trigger the granulator to look for derivative slices that refer to the full file, updating accordingly
2. Unsliced content & mixed-level integrations

1. Some information won't be present in the metadata of the derivative slice. For example, if I have a slice that processes by pies-item, I am still missing all of the PIES header information. I need some way to get this, and the full file is the common denominator

2. Most receivers are probably going to want to have the full file on hand to refer back to when initializing new systems or dealing with a proprietary system that still needs a full file input, even though other of their systems can handle L2 transactions. I anticipate, especially early on, that nearly everyone will subscribe to both level 1 and level 2 slices.

Like

Reply (/forums/post?tid=6020&ReplyPostID=6024&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
6/22/2020

We will need to document and reserve certain slice-metadata keywords. Maybe we should consider adding a sandpiper prefix like "sp:" to the ones we control (?).

But I'm still not convinced about a need for "sp:link", though. :)

Like

Reply (/forums/post?tid=6020&ReplyPostID=6025&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
6/22/2020

And maybe we need a prefix for local-only meta-data. These will not be synced.

local-only-metadata (/spaces/127/sandpiper/searchresults?keyword=%22local-only-metadata%22&searchtags=1)

Like

Reply (/forums/post?tid=6020&ReplyPostID=6026&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
6/23/2020

> On 6/22/2020 4:07 PM, Doug Winsby said:
> We will need to document and reserve certain slice-metadata keywords. Maybe we should consider adding a sandpiper prefix like "sp:" to the ones we control (?).

I would go so far as to say that the user should not be able to specify link systems -- they have to use one of the ones we supply. The link is the primary object that allows us to connect to the Auto Care VCdb and so on, for example. The link supplies a system name and a key field name, with a stored value that is the key field value being referenced. Any custom metadata the user wants to add has to go into the "Primary Actor" or "Secondary Actor" system.

For example, these are the link systems I have today:

System	KeyField	Examples	Description
Auto Care Branding	BrandOwner	XYYY	Auto Care Brand table brand owner entries
Auto Care Branding	Brand	XXYY	Auto Care Brand table brand entries
Auto Care Branding	SubBrand	XXXY	Auto Care Brand table subbrand entries
SWIFT	BIC	APGGDE10311	SWIFT Institute Business Identifier Code
NAPA Branding	MfrCode	12345	NAPA Brand ID entries
Primary Actor	Reference Code		Primary actor's internal codes
Secondary Actor	Reference Code		Secondary actor's internal codes
Auto Care VCdb	Version	2099-01-27	
Auto Care VCdb	Make	12	
Auto Care Padb	Version	2099-01-27	
Auto Care PCdb	Version	2099-01-27	
Auto Care PCdb	PartTerminology	12345	PartTerminologyID
Auto Care Qdb	Version	2099-01-27	
Root	SliceID	87446ffb-1bd0-4e19-a28d-ea995fa83939	Used to tie derivative slices to a root slice

In the XML this will look like:

```
<Slice id="be6c9e83-1007-4cf6-ac92-d1c3ece1c4d0" format="PIES" version="7.1" filename="AP
G_BAP_Wipers_PIES71_Full_2099-04-15.XML" type="full-file">
        <Description>Wiper Part Information</Description>
        <Links>
                <Link id="4bb75b11-754e-4367-937e-ffbeab316018" system="Primary Actor" ke
yfield="Reference Code" key="123456">Wipers</Link>
                <Link id="1b8c9d6b-96c8-4d9b-a251-c4b3e30d1c11" system="Secondary Actor"
keyfield="Reference Code" key="LLT">BAP WIPER ARMS AND BLADES</Link>
                <Link id="4223597c-d71e-48e6-98a8-9e4c3adafd66" system="Auto Care PCdb" k
eyfield="Version" key="2099-01-27">2099-01-27 Auto Care PCdb</Link>
                <Link id="693d8717-0c33-4f53-b6b7-912b76002b8f" system="Auto Care PAdb" k
eyfield="Version" key="2099-01-27">2099-01-27 Auto Care PAdb</Link>
        </Links>
</Slice>
```

Like

Reply (/forums/post?tid=6020&ReplyPostID=6027&SpaceID=127)   Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
6/23/2020

From a reference-implementation standpoint, `slice-metadata` is defined as:

```
CREATE TABLE IF NOT EXISTS "slice_metadata" (
  "slice_id" uuid REFERENCES "slices" ("id") ON DELETE CASCADE,
  "key"      text,
  "value"    text,
  PRIMARY KEY ("slice_id", "key")
);
```

Notice that we have a "key" and a "value". I would anticipate reference-versions such as:

```
"vcdb-version": "2020-06-26"
"pcdb-version": "2020-06-26"
```

I would recommend a **non-sync prefix** such as:

> ```
> "xx:pim-reference": "1b8c9d6b-96c8-4d9b-a251-c4b3e30d1c11"
> ```

The "xx:" prefix would indicate to the sync process not to deliver these slice-metadata key/values to the secondary database.

I'm also not sure what the use case for "make" would be for a slice. Are you thinking of L3 (which has not really been defined yet)?

The current L2 slice-types are:

> ```
> "aces-items"
> "asset-files"
> "pies-items"
> "pies-marketcopy"
> "pies-pricesheet"
> ```

Like

Reply (/forums/post?tid=6020&ReplyPostID=6032&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
6/24/2020

> **On 6/23/2020 5:46 PM, Doug Winsby said:**
>
> The "xx:" prefix would indicate to the sync process not to deliver these slice-metadata key/values to the secondary database.

I think that would work for internal-only metadata, though I think most of what we're discussing here is something that would still be shared (vcdb version etc.).

> **On 6/23/2020 5:46 PM, Doug Winsby said:**
>
> I'm also not sure what the use case for "make" would be for a slice. Are you thinking of L3 (which has not really been defined yet)?

You're right, we haven't defined that. I was just putting pieces in with the thought of future methods of granulating data, which we had planned to take on after 1.0 -- with Luke perhaps building a standard granulator reference implementation, perhaps with some novel ways of dividing slice data.

You had mentioned a week or two ago that defining these standard codes is a task we'll have to take on too, so I started my spreadsheet of possibilities. I think post-1.0 we could look at defining them deeper. It may also be that we want to leave the door open for a "custom" grain type that a granulator can further elucidate somehow.

> **On 6/23/2020 5:46 PM, Doug Winsby said:**
>
> From a reference-implementation standpoint, `slice-metadata` is defined as:
>
> ```
> CREATE TABLE IF NOT EXISTS "slice_metadata" (
>   "slice_id" uuid REFERENCES "slices" ("id") ON DELETE CASCADE,
>   "key"      text,
>   "value"    text,
>   PRIMARY KEY ("slice_id", "key")
> );
> ```
>
> Notice that we have a "key" and a "value". I would anticipate reference-versions such as:
>
> ```
> "vcdb-version": "2020-06-26"
> "pcdb-version": "2020-06-26"
> ```

I see now how the system -> Key field paradigm doesn't quite fit into this. The alternative then is to create the single field values like you mentioned here, which from a database standpoint rankles me a bit but with the number of values being so small it probably makes much more sense. I think we'll still want to tag these auto care, napa, etc. (e.g. "aces-vcdb-version" or similar), so that when there's an overlap between names used by different systems, we don't have to do any dancing. I'll reply to the sample plan post with my thoughts on that and how I'll be revising it.

Like

Reply (/forums/post?tid=6020&ReplyPostID=6033&SpaceID=127) Answer