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

Pools vs. Slices

Unsubscribe from this discussion
Subscribe to RSS (../../../../../../spaces-cf/forums/rss-space-posts.ashx?spaceID=127&topicID=5242&forumID=5044&key=rrer%2B1xDo%2BlxpC7jBRbM5w%3D%3D)

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
10/9/2019

Creating a new thread so we can discuss the pool and the slice, and what level they should exist at. I'll move one of my posts over here, from the reference data discussion, so we can go into it in more detail.

Like

Reply (/forums/post?tid=5242&ReplyPostID=5243&SpaceID=127)

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
10/9/2019

(Moved from reference data discussion)

In thinking more, this has helped me to change how I was thinking about the pool vs. a slice. It's given me a good way to approach the XML manifest as a way to solve some of this. Here's a proposal.

First, a pool should be one type of data. A single primary node should have multiple canonical pools, one for each type -- for example, one ACES application pool, one NAPA application pool, one PIES pool, etc. -- because the categorization, processing, and storage requirements for each of these will be vastly different.

The pool itself then explicitly states the system and subsystem that it references, as well as their versions. Autocare would be the system, and the subsystem is PCdb, VCdb, PAdb, Qdb, Branding, etc. The versions are their dates in this case; other systems would use other version schemes potentially. (How to handle schema versions -- I think in the same structure. Need to decide that)

The pool is actually a special kind of slice; a slice of all the data types, that is required to make that one special data type declaration.

Each pool then contains one or more slices. A slice can reference key values from the systems defined at the pool level, but at each level, only one *type* of reference can be made. For example, all of the first level slices might reference brand owner from the Autocare branding table, but they may not reference any other field, to ensure that they are a true A-B choice using the same kind of decision. There is a nice way to do this in the XML.

For an imaginary supplier Bob's Auto parts that wants to segment their data this way (note I'm only going down to detail in one of each slice below, presumably every one could go deeper):

1. Bob's Auto Parts ACES Applications
   1. Bob's Brakes
      1. ULTRA Brakes
      2. PERFORM Brakes
         1. Brake Friction
         2. Rotors
         3. Drums

The XML could define this:

```
<Pool DataCategory="Fitment" DataType="ACES">
   <Descriptions>
      <Description Type="Full">Bob's Auto Parts: ACES Applications</Description>
   </Descriptions>
   <System SystemID="Autocare">
      <SubSystem SystemID="VCdb" SystemVersion="9/27/2019" id="1" />
      <SubSystem SystemID="PCdb" SystemVersion="9/27/2019" id="2" />
      <SubSystem SystemID="Qdb" SystemVersion="9/27/2019" id="3" />
      <SubSystem SystemID="Brand Table" SystemVersion="10/1/2019" id="4" />
   </System>
   <Slices>
      <SystemLink LocalSystemID="4">Autocare Brand Table</SystemLink>
      <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD
(https://sandpiper.bapbrakes.com/AABBCCDD)">
         <Descriptions>
            <Description Type="Full">Bob's Auto Parts: Brakes</Description>
            <Description Type="Short">BAP Brakes</Description>
```

```
        </Descriptions>
        <Links>
          <LinkEntry LocalSystemID="4" KeyFieldID="BrandOwner" KeyFieldValue="XXXA">Bob's
Brakes</LinkEntry>
        </Links>
        <Slices>
          <SystemLink LocalSystemID="4" KeyFieldID="Brand"/>
          <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD/AA
(https://sandpiper.bapbrakes.com/AABBCCDD/AA)">
            <Descriptions>
              <Description Type="Full">ULTRA Braking Solutions</Description>
              <Description Type="Short">ULTRA</Description>
            </Descriptions>
            <Links>
              <LinkEntry KeyFieldValue="XXXB">ULTRA Brake Premium</LinkEntry>
              <LinkEntry KeyFieldValue="XXXC">ULTRA Brake Value</LinkEntry>
            </Links>
          </Slice>
          <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD/BB
(https://sandpiper.bapbrakes.com/AABBCCDD/BB)">
            <Descriptions>
              <Description Type="Full">PERFORM Braking Solutions</Description>
              <Description Type="Short">PERFORM</Description>
            </Descriptions>
            <Links>
              <LinkEntry KeyFieldValue="XXXD">PERFORM Brake</LinkEntry>
            </Links>
            <Slices>
              <SystemLink LocalSystemID="2" KeyFieldID="PartTerminologyID"/>
              <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD/BB/AA
(https://sandpiper.bapbrakes.com/AABBCCDD/BB/AA)">
                <Descriptions>
                  <Description Type="Full">Bob's PERFORM Brake Friction</Description>
                  <Description Type="Short">PERF Brake Friction</Description>
                </Descriptions>
                <Links>
                  <LinkEntry KeyFieldValue="1234">Disc Brake Pad</LinkEntry>
                  <LinkEntry KeyFieldValue="5678">Disc Brake Pad Set</LinkEntry>
                  <LinkEntry KeyFieldValue="91011">Electronic Wear Sensor</LinkEntry>
                </Links>
              </Slice>
              <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD/BB/BB
(https://sandpiper.bapbrakes.com/AABBCCDD/BB/BB)">
                <Descriptions>
                  <Description Type="Full">Bob's PERFORM Brake Rotors</Description>
                  <Description Type="Short">PERF Brake Rotors</Description>
                </Descriptions>
                <Links>
                  <LinkEntry KeyFieldValue="121314">Disc Brake Rotor</LinkEntry>
                  <LinkEntry KeyFieldValue="151617">Disc Brake Rotor and Hub
Assembly</LinkEntry>
                </Links>
              </Slice>
              <Slice uri="https://sandpiper.bapbrakes.com/AABBCCDD/BB/CC
(https://sandpiper.bapbrakes.com/AABBCCDD/BB/CC)">
                <Descriptions>
                  <Description Type="Full">Bob's PERFORM Brake Drums</Description>
                  <Description Type="Short">PERF Brake Drums</Description>
                </Descriptions>
                <Links>
                  <LinkEntry KeyFieldValue="181920">Brake Drum</LinkEntry>
                </Links>
              </Slice>
            </Slices>
          </Slice>
        </Slices>
      </Slice>
    </Slices>
  </Pool>
```

Like

Reply (/forums/post?tid=5242&ReplyPostID=5244&SpaceID=127) Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
10/9/2019

My thoughts: I could see the pool still needing to refer to the aggregate of all data. I wonder if instead we'd have a slice "Root" entity that is the only one allowed to have the database versioning and the data type indication.

Since a slice itself is a cut of existing data, it shouldn't re-define anything that's properly at the higher level, and affects other slices. You should be able to descend into a slice and know unambiguously that it will not conflict with or redefine other slices (note that this does not mean that it won't *overlap*, only that it won't alter your ability to interpret the data in any other place). If we create that root element, whatever it's called, the pool can retain its original meaning.

Like

Reply (/forums/post?tid=5242&ReplyPostID=5245&SpaceID=127) Answer

Doug Winsby (https://autocare.communifire.com/people/dougwinsby)
10/11/2019

I found it helpful to review our discussions under "The Slice (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5111/the-slice)" and "The DataObject (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5128/the-dataobject)". Here are a few things that seemed to gain consensus:

1. Slices represent a "syncable unit" (with a hash representing their contents).

2. Slices cannot be sub-divided, but can be combined for a subscriber.

3. DataObjects belong to one (or more) Slices.

4. DataObjects are immutable without state, history or "net-change" logic.

5. DataObject Granularity will be at "item-level" (unless non-item data like MarketCopy and PriceSheets).

Our discussions around "reference versioning (https://autocare.communifire.com/spaces/127/sandpiper/forums/api/5223/referencing-standard-dbs-and-versioning)" probably changed our thinking on #3 above. It looks like we might need a `data-object` to be assigned to just one `slice`. But we might want to assign a `data-object` to more than one `subscriber`.

So I propose adding a `channels` level between `subscribers` and `slices` to allow slices to be assigned to more than one subscriber. We had this concept before with a "subscriptions" level, but there was some thought we didn't need it, and I think "channels" is a better description). Maybe something like this:

> ```
> subscribers --> channels --> slices --> data-objects
> ```

This would allow a "WD" channel, for example, with several trading partners (subscribers) assigned to it. (**Question**: are there other channels that might have multiple subscribers other than "WD"?).

If "channels" have a different meaning than I'm proposing, we could go back to "subscriptions". The key, I think, is that we probably need some way to group `subscribers` if we are not allowing a `data-object` to be assigned to multiple `slices` (for reference versioning reasons).

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

Like

Reply (/forums/post?tid=5242&ReplyPostID=5257&SpaceID=127)   Answer

Krister Kittelson (https://autocare.communifire.com/people/krister-kittelson)
10/11/2019

An excellent point on the slice being indivisible at this point. My brain reverted to the earlier idea of having it nested, when doing the prototyping for this.

Like

Reply (/forums/post?tid=5242&ReplyPostID=5258&SpaceID=127)   Answer

Page 1 of 1 (5 items)