taxon description language syntax overhaul

feb 16 2021

definitions

movement - a block's position (coordinates of its centroid) changes

actuation - the block's children in the kinematic tree move, but the block itself does not

block - umbrella term referring to a motor, a tool, or a mechanism (formerly called "block", see change log below for more info)

changes:

- remove `buildEnvironment` property (maybe just for now)
- standardize `dimensions` in envelope such that `workEnvelope ::= (shape, dimensions, position)`
- rename the former top-level `blocks` property to `mechanisms`. objects that go in that are property are now known as `"mechanisms"`. "blocks" now refers collectively to anything that's part of the machine that's rendered as a box: `motors`, `mechanisms`, `tools`.
- remove the `componentType` property everywhere. in the case of motors and tools, the value was always `"motor"` or `"tool"` respectively anyway, so we can infer this by just looking at the top-property anyway. in the case of mechanisms (formerly blocks), we resolve this issue as described below.
- for mechanisms (formerly blocks), replace `componentType` with `mechanismType` which can be either: `linear`, `parallel`, `cross`, or `nonActuating` (perhaps deltaBot some day soon). this emphasizes that the top-level abstraction of the mechanism is *only* about how many motors it is driven by, and how many axes it actuates on. by definition:
  - a linear mechanism has one driving motor and actuates on one axis

    - a parallel mechanism has 2 or more driving motors and actuates on one axis
    - a cross mechanism has 2 driving motors and actuates on two axes
    - a non-actuating mechanism has 0 driving motors and actuates on 0 axes
    - ideally, we check `mechanismType`, `drivingMotors`, and actuationAxes to enforce this definition.
- in mechanism objects, rename `“axes”` to be `“actuationAxes”`
- in motor objects, remove `“drivenStages”`—now the pairing of motors and mechanisms (formerly called blocks or stages) is indicated only within `“drivingMotors”` within a mechanism object.
- remove explicit `kinematics` as a mechanism-level property. all linear and all parallel mechanisms are assumed to have `directDrive` kinematics. all cross mechanisms have `hBot`/`coreXY` kinematics (the kinematic equations are the same, but the physical properties might be different. to address this, we move hBot vs. coreXY into the attributes property). we now only use terms like "directDrive" in the compiler and these terms are never exposed to the programmer.
- remove `“platform”` and `“toolAssembly”` as mechanism-level properties; instead, these now go in the mechanism’s attributes property. again, this emphasizes that what we care about at the mechanism level is the input of motors and output of actuation.
- add new `metafeatures` property whose value is an object that contains any information about the machine that does not pertain to its volumetric or kinematic properties. as such, we move the formerly top-level `“vendorInfo”` and `“machineType”` properties in here. properties in the metafeature object are all optional, so we might as well add “version” as well.
- remove `connections` as a top-level property: now, blocks “own” their connections in which they are the parent (formerly called baseBlock) e.g.

```
“mechanisms”: [
   {
        “name”: “someParentMechanism”,
        “connections”: [
             {
                 “child”: “someChildMechanism”
                  …
             }
        ]
   }
```

```
]
```

- Condense connection syntax as shown, e.g. from:

```
{
    "baseBlockName": "leadscrew motor a",
    "baseBlockFace": "-y",
    "baseBlockEnd": "0",
    "addBlockName": "z leadscrews",
    "addBlockFace": "+y",
    "addBlockEnd": "+z"
},
```

to:

```
{
    // implicit: baseBlock is the block that owns this connection
    “child: "y leadscrews",
    “parentPoint": "-y.center",
    “childPoint": "+y.+z"
}
```

where `“basePoint”` and `“addPoint”` have the syntax “F.P” where F is in [+/-x, +/-y, +/-z]