Loading Attribute Values

You may need to load the same values that Structure shows on the Structure Board, especially if it's a total value, progress value or other Structure-specific value. This is done via `StructureAttributeService`.

About Attributes

One of the core concepts in Structure is the Attribute abstraction. An attribute is something that can provide a value of specific type and meaning for any row in a forest.

For example, a "Summary" attribute would produce the value of Summary field for issues, the name of a folder for folders and a person's full name for users. Some attributes may be applicable only to certain item types and would provide empty value for all other items.

Besides item-based attributes, which provide values that depend only on the item in the forest, there are forest-based attributes, which are calculated based on the whole forest and items in it.

Forests and Attributes are two main concepts that make up the Structure grid. Looking at the Structure Board, you see Forest in the vertical direction – rows and hierarchy are taken from Forest, and you see Attributes in the horizontal direction – all columns load Attributes from the server and display those values.

General Approach to Loading Values

Let's assume that, after Reading Structure Content, you have `StructureComponents` instance and an instance of `ForestSpec` for a forest. We can read a number of attributes for a number of rows by going to `StructureAttributeService.`

1. Figure out which Attributes do you need

The service accepts multiple attribute specs in one request. If you need several attributes calculated – it's better to do that in one request.

```
List<AttributeSpec<?>> attributeSet = new ArrayList<>();
attributeSet.add(CoreAttributeSpecs.KEY);
attributeSet.add(CoreAttributeSpecs.SUMMARY);
attributeSet.add(CoreAttributeSpecs.TOTAL_REMAINING_ESTIMATE);
```

`CoreAttributeSpecs` class and its parent class, `SharedAttributeSpecs`, contain some of the most popular attributes.

It's likely that you'll need to build you own attribute specification. For example, to address a numeric JIRA custom field and calculate total of that field based on sub-issues, you'll need the following.

```
AttributeSpec<Number> customField =
  AttributeSpecBuilder.create("customfield", ValueFormat.NUMBER).params().set("fieldId", 10000).build();

AttributeSpec<Number> customFieldTotal =
  AttributeSpecBuilder.create(CoreAttributeSpecs.Id.SUM, ValueFormat.NUMBER).params().setAttribute(customField).
build();

attributeSet.add(customFieldTotal);
```

2. Figure out which Rows do you need to calculate the Attributes for

For example, this could be all rows in that structure.

```
LongList rows = myStructureComponents.getForestService().getForestSource(forestSpec).getLatest().getForest().
getRows();
```

If you need to create a `LongList` manually, use `LongArray` implementation.

3. Call `StructureAttributeService`

This service calculates a matrix of values for each row and attribute you specify.

```
RowValues values = myStructureComponents.getAttributeService().getAttributeValues(forestSpec, rows,
attributeSet);
```

There is a variation of `getAttributeValues()` method that accepts a `Forest`, rather than `ForestSpec`. It is recommended to use the variant that accepts `ForestSpec` whenever possible, because that variant uses caching.

4. Read out the result

The returned object contains values for all pairs of requested row and requested attribute.

```
for (LongIterator ii : rows) {
  String key = values.get(ii.value(), CoreAttributeSpecs.KEY);
  Number total = values.get(ii.value(), customFieldTotal);
  ...
}
```