Query for progress on an epic

What

I would like to get data about the total estimate and progress of epics in a reporting table so I can create my own reports. How can I get this kind of data about epics through the API?

How

This endpoint was introduced in 13.2, Summer 2013. Please check Support Center -> About to see if you are on this release or later.

The following uses the `query.v1` endpoint to obtain data about epics. While there are other approaches, query.v1 makes it easy to write a readable query and get JSON results.

Getting Started

- Have an HTTP Client.
- Obtain an API token.

from

The `from` parameter should reflect our desire to use the *Epic* asset type.

```from: Epic```

If our query only has a `from` parameter, we get a default set of attributes. This default set does not include any results from the workitems it contains so we need to use `select` parameters to specify what we want. Most attribute definitions can be found by a query to `meta.v1`. To see the attributes available for *Epic*, perform the following query.

```<Server Base URI>/meta.v1/Epic?xsl=api.xsl```

The result will resemble the following, except with many more attributes.

```
Timebox derives from BaseAsset
* Name : Text
* Scope : Relation to Scope — reciprocal of Workitems
  Description : LongText
  Status : Relation to EpicStatus — reciprocal of Epics
```
Simple Attribute: Name

We can use any of the attributes directly in the `select`. Let's add `Name`.

```
from: Epic
select:
- Name
```

Multi-Relation: Names of Children

We can also construct a complex attribute using the `attribute definition syntax`. The `Workitems` under an `Epic` are known as `Subs`. We can get the `Name` from each of those `Workitems`. Let's add that to the `select`.

```
from: Epic
select:
- Name
- Subs.Name
```

Multi-Relation: Names of All Descendants

The `Subs` only include the `Workitems` where the `Epic` is a direct child. We can also get all the descendants; in other words, recursively descent the `Epics` and get the `Subs` for each one, adding it to the set of results. For that, we use `SubsAndDown`.

```
from: Epic
select:
- Name
- SubsAndDown.Name
```

Downcasting: Only Primary Workitems

The `Subs` and `SubsAndDown` relationships include other `Epics`. We can use `downcasting` to tell the query to just return `PrimaryWorkitems` which effectively filters out `Epics`. It also allows us to access the attributes that are specific to a subtype. With a downcast, we can now `select` the `Estimate` attribute.

```
from: Epic
select:
- Name
- SubsAndDown:PrimaryWorkitem.Estimate
```
Filtering: Only Closed Workitems

We can also narrow the children using a filter to select those that are closed.

```plaintext
from: Epic
select:
  - Name
  - SubsAndDown:PrimaryWorkitem[AssetState='Closed'].Estimate
```

Aggregation: Sum it up for me

Now we can put it all together and sum up the values we need using @Sum. By design, queries for normal attributes will automatically filter deleted and templates, but aggregates have no automatic filtering. Therefore, it is necessary to have a filter even when we want both open and closed children.

```plaintext
from: Epic
select:
  - Name
  - SubsAndDown:PrimaryWorkitem[AssetState='Closed'].Estimate.@Sum
  - SubsAndDown:PrimaryWorkitem[AssetState!='Dead'].Estimate.@Sum
```

where

If our query does not have a where or filter parameter, the results will include every Epic. Let’s look at some options for reducing the result set.

Unique Match: ID

If we know the Number for a specific Epic, we can use it in the where to make sure we get a single asset in the result set.

```plaintext
from: Epic
select:
  - Name
  - SubsAndDown:PrimaryWorkitem[AssetState='Closed'].Estimate.@Sum
  - SubsAndDown:PrimaryWorkitem[AssetState!='Dead'].Estimate.@Sum
where:
  ID: E-01002
```

Filter: Inequality

Alternatively, we might want only those Epics that have some estimates from which to calculate a percent complete. For that we use a filter to model inequality.

```plaintext
from: Epic
select:
```
In order to execute the query, submit an HTTP POST with the query as the body to query.v1.