Bulk Data and Directories

Background

Bulk data is a recent addition to the FHIR related specifications and has many uses. Primarily its for out of band data extraction for distribution.

It is very useful in extraction for processing into Bulk data systems for analytics purposes, performance measure reporting, or sharing content, or govt extracts etc.

There are several basic parts to FHIR's bulk data extract

- The scope of data selection
- The format of the bulk data extract (nd-json) *
- The async operation request and status tracking (as these processes may take significant processing time) *
- Retrieval of the completed export
- Closing/cleanup of the completed export *

*(The items marked with * can be considered quite mature, and unlikely to change from this point)*

The draft specification for this functionality can be found at:

> https://github.com/smart-on-fhir/fhir-bulk-data-docs
>
> https://github.com/smart-on-fhir/fhir-bulk-data-docs/blob/master/export.md

As this has matured through connectathon experience and essentially agreed upon, the formal HL7 artifacts are being produced, and can be found here:

> http://build.fhir.org/ig/HL7/bulk-data-export

And the community discussions and questions around this draft specification are here:

> https://chat.fhir.org/#narrow/stream/179250-bulk-data

This draft specification has been quite well exercised over the last 18 months since the January 2018 FHIR Connectathon, and has been implemented by Epic, Cerner, CDC (USA), SMART Health IT test program (Harvard Children's Hospital), Care Evolution, Health Intersections, HSPC/Intermountain Healthcare and likely others.

Josh Mandel and Dan Gottlieb created a presentation that walks through all this too:

> https://docs.google.com/presentation/d/14ZHmam9hwz6-SsCG1YqUIQnJ56bvSqEatebltgEVR6c/present?ueb=true&slide=id.g45ba049ea1_0_140

Healthcare Directory Bulk Data Specifics

The scope of the data selection

For the directory bulk data extraction, to request an entire copy of all content in the directory, the scope selection can be defined at the top level, and just specifying that we would like to retrieve all content for the specified resource types from the base of the FHIR server.

```
GET [base]/$export?_type=Organization,Location,Practitioner,PractitionerRole,HealthcareService,
VerificationResult, ...
```

A healthcare directory may curate such an extract on a nightly process, and just return this without needing to scan the live system, and the value returned in the **`transactionTime`** in the result should contain the timestamp at which this was generated (including timezone information), and that should be used in a subsequent call to retrieve changes since this point in time.

Once a system has a complete set of data, it is usually more efficient to ask for changes since a point in time, in which case the request should use the value above (**`transactionTime`**) to update the local directory.

```
GET [base]/$export?_type=Organization,Location,Practitioner, ... &_since=[transactionTime]
```

This behaves just the same as the initial request, with the exception of the content.

We would expect that this is more likely to return the current information, rather than from a pre-generated snapshot, as the transactionTime could be anything.

> ***Note:*** *The current bulk data handling specification does not handle deleted items, and the recommendation is that periodically a complete download should be done to check for "gaps" to reconcile the deletions (which could also be due to security changes), however content shouldn't usually be "deleted" it should be marked as inactive, or end dated.*

Proposal: *Include deletions bundle(s) for each resource type to report the deletions (when using the _since parameter) which would be in a new property "deletions" in the process output, as demonstrated in the status tracking output section below. This bundle would have a type of "collection", and each entry would be as per a deleted item in a history*

```
<entry>

  <!-- no resource included for a delete -->
  <request>
    <method value="DELETE"/>
    <url value="PractitionerRole/[id]"/>
  </request>
  <!-- response carries the instant the server processed the delete -->
  <response>
    <lastModified value="2014-08-20T11:05:34.174Z"/>
  </response>
</entry>
```

The total in the bundle will just be the count of deletions in the file, the total in the operation result will indicate the number of deletion bundles in the ndjson (same as the other types).

If the caller doesn't want to use the deletions, they can just ignore the files in the output, and not download those specific files.

List defined subsets

The previous sections are all that is defined by the FHIR Bulk Data extract specification, however we may choose to implement an additional parameter to this operation to permit the selection to also filter to resources that are included in a specified list resource. The approach is similar to the same capability defined by FHIR http://hl7.org/fhir/search.html#list

This could be used by client applications such as a Primary Care System that wanted to only periodically update using this technique, but only with resources that they currently have loaded in their "local directory" - internal black book, which were cached there from previous searches to the system.

```
GET [base]/$export?_type=Organization,Location,Practitioner,PractitionerRole,HealthcareService&_list=List/45
```

In this example the Primary Care System would be responsible for keeping `List/45` up to date with what it is tracking, and a national service may decide that permitting this List resource management is too much overhead, however local enterprise directories may support this type of functionality.

Arbitrary subsets of data

The current bulk data export operations use the Group resource to define the set of data related to a Patient, and at present there is no definition for this to be done in the directory space, apart from at the resource type level. This is possible with search and subscriptions (which leverage search) by using search parameters on the resource types and setting the parameters of interest.

When defining a subset of data, consideration should be given to what happens when data is changed such that it no longer is within the context of the conditions.

One possibility would be to use a bundle of searches where each type has it's own search parameters, another is to use a GraphDefinition resource.

This functionality should be the subject of a connectathon to determine practical solutions.

One possibility could be to leverage the List functionality described above to maintain a state of what *was* included in a previous content, however this obviously incurs additional overhead on the part of the server, and for many systems - especially those at scale like a national system - this is likely not practical.

Format of the bulk data extract

The bulk extract format is always an nd-json file (new-line delimited json), and each file can only contain 1 resource type in it, but can be spread across multiple files, with either a size limit or count limit imposed by the extracting system, not the requestor.

The list of these files will be returned in the Complete status output, as described in the standard Bulk Data documentation.

Starting the extract

There are 2 options for starting the extract, one that is a single operation specifying all the content, and the other to be for a specific type only. These were both covered in the selecting the scope section above.

Here I will only document the use of the global export, as an initial request.

The initial request:

```
GET [base]/$export?_type=Organization,Location,Practitioner,PractitionerRole,HealthcareService
with headers:
Accept: application/fhir+json
Authentication: Bearer [bearer token]
Prefer: respond-async
```

This will return either:

- a status 4XX or 5XX with an `OperationOutcome` resource body if the request fails,
- or a status 202 Accepted when successful, with a `Content-Location` header with an absolute URI for subsequent status requests, and optionally an `OperationOutcome` in the resource body if desired

Example Content-Location: http://example.org/status-tracking/request-123 (note that this is not necessarily a FHIR endpoint, and isn't a true FHIR resource)

Tracking the status of the extract

After a bulk data request has been started, the client MAY poll the URI provided in the `Content-Location` header.

```
GET http://example.org/status-tracking/request-123
```

This will return:

- HTTP Status Code of `202 Accepted` when still in progress (and no body returned)
- HTTP status code of 5XX when a fatal error occurs, and an `OperationOutcome` in json format for the body with the detail of the error (Note this is a fatal error in processing, not some error encountered while processing files - a complete extract can contain errors)
- HTTP status of `200 OK` when the processing is complete, and the result is a json object as noted in the specification (and an example included below)

Refer to the build data specification for details of the completion event:

https://github.com/smart-on-fhir/fhir-bulk-data-docs/blob/master/export.md#response---complete-status

```
{
   "transactionTime": "[instant]",
   "request" : "[base]/$export?_type=Organization,Location,Practitioner,PractitionerRole,HealthcareService",
   "requiresAccessToken" : true,
   "output" : [{
     "type" : "Practitioner",
     "url" : "http://serverpath2/practitioner_file_1.ndjson",
     "count" : 10000
   },{
     "type" : "Practitioner",
     "url" : "http://serverpath2/practitioner_file_2.ndjson",
     "count" : 3017
   },{
     "type" : "Location",
     "url" : "http://serverpath2/location_file_1.ndjson",
     "count" : 4182
   }],

// Note that this deletions property is a proposal, not part of the bulk data spec.
   "deletions" : [{
     "type" : "PractitionerRole",
     "url" : "http://serverpath2/practitionerrole_deletions_1.ndjson", // the bundle will include the total
number of deletions in the file
     "count" : 23 // this is the number of bundles in the file, not the number of resources deleted
   }],

   "error" : [{
     "type" : "OperationOutcome",
     "url" : "http://serverpath2/err_file_1.ndjson",
     "count" : 439
   }]
 }
```

Retrieving the complete extract

Once the tracking of the extract returns a **200 OK** completed status, the body of the result will include the list of prepared files that you can download.

Then each of these URLs can be downloaded by a simple get, ensuring to pass the Bearer token if the result indicates **requiresAccessToken = true**

While downloading, also recommend including the header Accept-Encoding: gzip to compress the content as it comes down.

```
GET http://serverpath2/location_file_1.ndjson
```

(Note: our implementation will probably always gzip encode the content - as we are likely to store the processing files gzip encoded to save space in the storage system)

Once you have downloaded all the files you need, you should tell the server to cleanup, which is detailed next.

Finishing the extract

This is the simplest part of the process, and that is just calling **DELETE** on the status tracking URL.

```
DELETE http://example.org/status-tracking/request-123
```

This then tells the server that we are all finished with the data, and it can be deleted/cleaned up. The server may also include some time based limits where it may only keep it for a set period of time before it automatically cleans it up.

Subscriptions

A close relative to the bulk data extract is the subscriptions content - and how these will work in the context of Bulk Directory exchanges needs further experimentation and connectathon experiences.

These could be setup to monitor the directory for realtime changes to resources of interest, and are defined through the use of search parameters

The "urgent notifications" channel is yet to be defined, but should enable specific actions such as license suspensions/revocations.