3.2.0 Extended Operations on the RESTful API

FHIR Infrastructure (http://www.hl7.org/Special/committees/fiwg/index.cfm) Work Group	Maturity Level (versions.html#maturity): Normative	Standards Status (versions.html#std-process): Normative (versions.html#std-process)

This page has been approved as part of an ANSI (https://www.ansi.org/) standard. See the Infrastructure (ansi-infrastructure.html) Package for further details.

The RESTful API (http.html) defines a set of common interactions (read, update, search, etc.) performed on a repository of typed resources. These interactions follow the RESTful paradigm of managing state by **C**reate/**R**ead/**U**pdate/**D**elete actions on a set of identified resources. While this approach solves many use cases, there is some functionality that can be met more efficiently using an RPC-like paradigm, where named operations are performed with inputs and outputs (**E**xecute).

Operations are appropriately used where:

- the server needs to play an active role in formulating the content of the response, not merely return existing information
- the intended purpose is to cause side effects such as the modification of existing resources or creation of new resources, or other changes out of scope of the RESTful interface (e.g. merging patient records)
- The task involves business rules to be enforced across multiple different resources and/or
- the task involves updating several resources in a coordinated manner (note: this can also be done by a Transaction (http.html#transaction), but an operation can be more focused on the task

This specification describes a lightweight operation framework that seamlessly extends the RESTful API. The framework covers both how to execute such an operation (this page) and how to define an operation (operationdefinition.html).

Operations have the following general properties:

- Each operation has a name
- Each operation has a list of 'in' and 'out' parameters
- Parameters are either resources, data types, or search parameters
- Operations are subject to the same security constraints and requirements as the RESTful API
- The URIs for the operation end-points are based on the existing RESTful API address scheme
- Operations may make use of all types of resources existing in the repository
- Operations may be performed on a specific resource, a resource type, or a whole system

3.2.0.1 Executing an Operation

Operations are executed using a URL derived from the FHIR endpoint, where the name of the operation is prefixed by a "dollar sign" ('$') character. For example:

```
POST http://fhir.someserver.org/fhir/Patient/1/$everything
```

When an operation has affectsState = false, and the parameters are all primitive data types with no extensions (as is the case with the example above), it may be invoked using GET as well. (Note: A HEAD request can also be used - see Support for HEAD (http.html#head)).

Operations can be invoked on four types of FHIR endpoints:

- The "base" FHIR service endpoint (e.g. http://fhir.someserver.org/fhir): These are operations that operate on the full scale of the server. For example, "return me all extensions known by this server"
- A Resource type (e.g. http://fhir.someserver.org/fhir/Patient): These are operations that operate across all instances of a given resource type
- A Resource instance (e.g. http://fhir.someserver.org/fhir/Patient/1): These are operations that involve only a single instance of a Resource, like the $everything operation above does

The body of the invocation contains a special infrastructure resource called Parameters (parameters.html), which represents a collection of named parameters as <key,value> pairs, where the value may be any primitive or complex datatype or even a full Resource. It may also include strings formatted as search parameter types.

Upon completion, the operation returns another `Parameters` resource, containing one or more output parameters. This means that a FHIR operation can take a set of zero or more parameters *in* and return a set of zero or more result parameters *out*. Both the body of the POST and the returned result are always a Resource.

Operations may be invoked using a `GET`, with parameters as HTTP URL parameters, if:

1. there are only simple input parameters - i.e. no complex datatypes like 'Identifier' or 'Reference', and
2. and the operation does not affect the state of the server (operationdefinition-definitions.html#OperationDefinition.affectsState)

If there is a single output parameter named 'return' then the response MAY be the resource that is the return value, with no Parameters resource. These kinds of usage are discussed further below.

If the response is a Bundle (bundle.html), the correct Bundle.type (bundle-definitions.html#Bundle.type) is 'collection (codesystem-bundle-type.html#bundle-type-collection)', unless it has search semantics (http.html#search), such as matching resource counts, and page links (next etc) (http.html#paging).

3.2.0.1.1 Operations with no parameters

Executing operations without any parameters is a special case. For an operation that doesn't cause any state change, the operation is invoked in a straight forward fashion:

```
GET [base]/Composition/example/$document
```

For operations that call state changes, they must be invoked by a POST. There is no parameters resource in this case because a parameters resource cannot be empty. So the operation is invoked with a POST with an empty body:

```
POST [base]/Claim/example/$submit
Content-Length: 0
```

3.2.0.2 FHIR defined Operations

See the list of defined operations (operationslist.html).

3.2.0.3 Implementation Defined Operations

Implementations are able to define their own operations in addition to those defined here. Name clashes between operations defined by different implementers can be resolved by the use of the server's Capability Statement (capabilitystatement.html).

Also, the definition of these or additional run time operations does not prevent the use of other kinds of operations that are not dependent on and/or not integrated with the RESTful API, provided that their addressing scheme does not clash with the scheme defined here.

3.2.0.4 Defining an Operation

Each Operation is defined by:

- A context for the Operation - *system*, *resource type*, or *resource instance*
- A name for the Operation
- A list of parameters along with their definitions

For each parameter, the following information is needed:

- Name - the name of the parameter. For implementer convenience, the name should be a valid token in common implementation languages
- Use - In | Out | Both
- Type - a data type or a Resource type
- Search Type - for string search parameters, what kind of search parameter they are (and what kind of modifiers they have)
- Profile - a StructureDefinition (structuredefinition.html) that describes additional restrictions about the parameter - only used if the parameter type is a resource or data type
- Documentation - a description of the parameter's use
- (Optional) Search Type - if the type is a string, and the parameter is being used like a search parameter, which kind of search type applies

Parameters may be nested into multi-part parameters. Each part has the same information as a parameter, except for use, which is taken from the parameter it is part of.

The resource Operation Definition (operationdefinition.html) is used to provide a computable definition of the Operation.

3.2.0.5 Extending an Operation

Implementations are able to extend an operation by defining new named parameters. Implementations can publish their own extended definitions using the Operation Definition (operationdefinition.html) resource, and this variant definition can use OperationDefinition.base to refer to the underlying definition.

Note that the FHIR specification will never define any parameter names starting with "x-".

3.2.0.6 Executing an Operation Synchronously

Operations are typically executed synchronously: a client sends a request to a server that includes the operation's *in* parameters and the server replies with the operation's *out* parameters.

The URL for an operation end-point depends on its context:

- system: the URL is [base]/$[name]
- resource type: the URL is [base]/[type]/$[name]
- resource instance: the URL is [base]/[type]/[id]/$[name]

3.2.0.6.1 Operation Request

An operation is generally invoked by performing an HTTP POST to the operation's end-point. The submitted content is the special Parameters (parameters.html) format (the "in" parameters) - a list of named parameters. For an example, see the value set expansion request example (op-example-request.html). Note that when parameters have a search type, the search modifiers are available and are used on the parameter name in the Parameters resource (e.g. "code:in").

Note that the same arrangement as for the RESTful interface applies with respect to content types (http.html#mime-type).

If all the parameters for the operation are primitive types (datatypes.html#primitive) and the operation has `affectsState` (operationdefinition-definitions.html#OperationDefinition.affectsState) = false, the operation may be invoked by performing an HTTP GET operation where all of the values of the parameters are appended to the URL in the search portion of the URL (e.g. after the '?' character). Servers SHALL support this method of invocation. E.g.

```
GET [base]/ValueSet/$expand?url=http://hl7.org/fhir/ValueSet/body-site&filter=abdo
```

When using the HTTP GET operation, if there is a repeating parameter for the extended operation the values for that parameter are repeated by repeating the named parameter. E.g. Observation $stats statistic parameter

```
GET [base]/Observation/$stats?subject=Patient/123&code=55284-4&system=http://loinc.org&duration=1&statistic=average&statistic=min&statistic=max&statistic=count
```

If, when invoking the operation, there is exactly one input parameter of type Resource (irrespective of whether other possible parameters are defined), that the operation can also be executed by a POST with that resource as the body of the request (and no parameters on the url).

Servers MAY choose to support submission of the parameters represented in multi-part/form-data (https://www.ietf.org/rfc/rfc2388.txt) format as well, which can be useful when testing an operation using HTML forms.

3.2.0.6.2 Operation Response

If an operation succeeds, an HTTP Status success code is returned. This will usually be a 2xx code, though it may also be a 303 See Other. Other kinds of 3xx codes should be understood to indicate that the operation did not proceed, and the client will need to re-issue the operation if it can perform the redirection (e.g. may get redirected to an authentication step). User agents should note that servers may issue redirects, etc. to authenticate the client in response to an operation request. An HTTP status code of 4xx or 5xx indicates an error, and an OperationOutcome (operationoutcome.html) SHOULD be returned with details.

In general, an operation response uses the same Parameters (parameters.html) format whether there is only one or there are multiple named *out* parameters.

If there is only one *out* parameter, which is a Resource with the parameter name "return" then the parameter format is not used, and the response is simply the resource itself.

The result of an operation is subject to content negotiation like any other interaction (http.html#mime-type). Specifically, if the returned resource is a Binary, the response SHALL behave in the same manner as if a 'read' operation had been performed on the resource. I.e. The content will be returned as either a FHIR resourse with base64-encoded content or as a raw binary, depending on the Accept header specified when invoking the operation (see Serving Binary Resources using the RESTful API (binary.html#rest)).

The resources that are returned by the operation may be retained and made available in the resource repository on the operation server. In that case, the server will provide the identity of the resource in the returned resources. When resources that are not persisted are returned in the response, they will have no `id` property.

3.2.0.7 Executing an Operation Asynchronously

Use the standard RESTful API Asynchronous pattern (async.html) to execute operations asynchronously.