DUKE UNIVERSITY LIBRARIES

Table of Contents

DUKE UNIVERSITY LIBRARIES

Table of Contents

API Methods Documentation

Binding Method

The repository is modeled in a Resource Oriented Architecture. All repository entities are resources, and are thus available and manipulated through RESTful interfaces. The API Methods listed below are RESTful web services. Details about each of these web services and how they are addressable are available in the Bindings section for each method.

The repository has a base URI. It is referred to throughout the bindings sections as repositoryURI.

Error codes and conditions are not yet addressed in the binding.

Item (General)

Function: CreateItem

Summary:

This function creates an item in the repository and returns a persistent identifier for that item.

Parameters:

- **Type** (string, required): the type of the item to be created in the repository.

Returns:

A persistent identifier for the item that was created in the repository.

Exceptional Conditions:

- **NotSupported** the underlying repository does not support this type of request.

Side Effects:

An item is created in the repository.

Rationale:

Any administrative tool for managing a repository will need to create items in the repository.

Notes:

Presumably, the item that is created in the repository is a skeletal record. In other words, it does not yet contain descriptive metadata, nor does it contain components, nor does it have relationships with other items in the repository.

In order to govern how the administrative tool continues to interact with that item, a repository may implement the creation of items differently depending on type. For instance, an image item may allow the administrative tool to associate image components with it, and a TEI item may not allow the administrative tool to edit the item's descriptive metadata. For repository implementations that do not make the distinction by item type, the type that is passed as an argument to this function may be disregarded.

DUKE UNIVERSITY LIBRARIES

Binding:

Request:

URI: http://{repositoryURI}/items
HTTP Method: POST
HTTP Parameters: type: the system type of the item to be created

Response:

HTTP Response Code: 201 Created
HTTP Location: the URI of the newly created item, eg. http://{repositoryURI}/items/duke-1; note that URI includes the permanent item identifier for the newly created item

---

Function: GetItem

Summary:

This function returns a representation of an item, in a given schema. The default schema for representing the return value is METS.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **Schema** (string, optional): the schema to return the requested item in; the default is METS

Returns:

A representation of the item in the requested (or default) schema.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **SchemaNotSupported**: the underlying repository does not implement returning a representation of the item in the requested schema
- **ItemNotFound**: the item specified by the ItemId was not found in the repository

Side Effects:

None

DUKE UNIVERSITY LIBRARIES

Rationale:

An administrative tool may be able to understand a complete representation of an item which includes the descriptive, administrative and structural metadata for the item. This function may also be useful for consumption by a discovery interface, parsing by an indexing engine, or harvesting by an external system.

Notes:

METS is the most acceptable standard for representing the descriptive, administrative and structural metadata for an item in a digital repository. And, so this function may be implemented by a repository and considered acceptable if METS is the only schema that is implemented. However, there are no requirements or restrictions on the schema that can be implemented.

This function may also suffice as an export mechanism for the repository. Although METS is the LOC-approved schema for representing items in a repository, this function may also implement schema that are native to the repository, such as FOXML for Fedora.

This function may also return a representation of all of the services that the item provides. In this case, the returned representation should include full URIs for all referenced resources.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}
HTTP Method: GET
HTTP Parameters: schema: (optional) the return schema of the response, default is METS, other conceived schema: Services, FOXML

Response:

An example response when METS is requested as schema:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<mets xmlns:duke="http://library.duke.edu/metadata/dukecore" xmlns:dcterms="http://p
  <metsHdr/>
  <dmdSec ID="R0277">
    <mdWrap OTHERMDTYPE="Duke Core -- Advertising" MDTYPE="OTHER">
      <xmlData>
        <dc:type>Advertisements</dc:type>

<dc:title duke:role="Headline">more efficient... in miniature </dc:title>
        <dc:date duke:role="Year">1945</dc:date>
```

```
<dc:subject duke:role="Company">Tung-Sol Electronic Tubes</dc:subject>
<dc:subject duke:role="Product">Radio Tubes</dc:subject>
<dc:source duke:role="Publication">Time</dc:source>
<dc:source duke:role="Publication Type">magazine</dc:source>
<dcterms:extent duke:role="Number of Pages">1</dcterms:extent>
<dc:audience duke:role="Target Audience">Consumer</dc:audience>
<dc:subject>Radio--Radio Tubes</dc:subject>


<duke:category>radio</duke:category>

        <duke:category>1940-1945</duke:category>
        <duke:collection>adaccess</duke:collection>
      </xmlData>
    </mdWrap>
  </dmdSec>
  <fileSec>
    <fileGrp USE="DEFAULT" ID="R0277">
      <file USE="THUMBNAIL" ID="R0277-thm">

        <FLocat xlink:href="http://library.duke.edu/digitalcollections/images/adacce
      </file>
      <file USE="MEDIUM" ID="R0277-med">
        <FLocat xlink:href="http://library.duke.edu/digitalcollections/images/adacce
      </file>
      <file USE="LARGE" ID="R0277-lrg">
        <FLocat xlink:href="http://library.duke.edu/digitalcollections/images/adacce
      </file>
    </fileGrp>

  </fileSec>
  <structMap>
    <div LABEL="1" TYPE="Page" ID="R0277" ORDER="1">
      <fptr fileID="R0277-thm"/>
      <fptr fileID="R0277-med"/>
      <fptr fileID="R0277-lrg"/>
    </div>
  </structMap>
</mets>
```

An example response when Services is the schema:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<item>
  <services>
    <service name="ItemInfo" label="Item Information">http://{repositoryURI}/items/{
    <service name="Type" label="Item Type">http://{repositoryURI}/items/{item-id}/ty
```

```
    <service name="Status" label="Item Status">http://{repositoryURI}/items/{item-id
    <service name="Relationships" label="Item Relationships">http://{repositoryURI}/
    <service name="DMR" label="Descriptive Metadata Record">http://{repositoryURI}/i
    <service name="MetadataForm" label="Metadata Form">http://{repositoryURI}/items/
    <service name="MAP" label="Metadata Application Profile">http://{repositoryURI}/
    <service name="Components" label="Components">http://{repositoryURI}/items/{item
</item>
```

Function: GetType

Summary:

This function returns the type of the specified item in the repository

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

This function returns the type of the specified item in the repository.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the item specified by the ItemId was not found in the repository

Side Effects:

None

Rationale:

Administrative tool may want to display the type of the item. The administrative tool may also have some logic to understand how to treat different item types differently. Discovery interfaces may also want to make display decisions that are informed by the type of item they are delivering.

Notes:

Binding:

DUKE UNIVERSITY LIBRARIES

Request:

URI: http://{repositoryURI}/items/{item-id}/type
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response returning item type:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<item>
  <itemType>Image</itemType>
</item>
```

Function: GetStatus

Summary:

Gets the status of the item in the underlying repository.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

This function returns the item status of the item in the underlying repository

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

DUKE UNIVERSITY LIBRARIES

Rationale:

An administrative tool will want to display the status of an item. Discovery interfaces may presumably make display decisions based on the status of an item. Indexing tools may make indexing decisions based on the status of an item.

Notes:

There are no restrictions or requirements on the definitions of statuses in the underlying repository. Some suggested statuses are Incomplete, Complete, and Published.

Bindings:

Request:

URI: http://{repositoryURI}/items/{item-id}/status
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response returning item status:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<item>
  <itemStatus>Published</itemStatus>
</item>
```

Function: SetStatus

Summary:

This function sets the status for an item in the digital repository.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **Status** (string, required): the status to set for the item in the repository
- **OverrideValidation** (boolean, optional, default=false): whether to override validation when setting the status

Returns:

Returns a message containing the item's id, the item's status, a return code specifying the success or failure of the request, and possibly a human readable success/failure message.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

If applicable to underlying repository implementation, item validation executed within the repository. The status of the item may be changed.

Rationale:

Administrative tool users will need the ability to change the status of an item in order to publish it, or remove it from the published domain. There may be a need, either programmatically, or from the administrative tool itself, to override validation in order to publish an item that does not meet all validation criteria.

Notes:

Repositories that implement this function may, or may not, have the ability to automate the validation process. In those repositories, the implementation of this method would disregard the OverrideValidation parameter.

Presumably, item will be assigned a status on initial item creation. Therefore, all invocations of the SetStatus method will be updates of status, not creation of new status resource, so PUT method will be used.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/status
HTTP Method: PUT
HTTP Parameters: status: (required) {Incomplete, Complete, Published}

overrideValidation: (optional) {yes, no}

Response:

HTTP Response Code: 200 OK

An example response returning successful message:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>00</responseCode>
  <responseMessage>Status Updated Successfully</itemStatus>
</response>
```

An example response returning error:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>01</responseCode>
  <responseMessage>Unable to set status, item failed validation</responseMessage>
</response>
```

Function: GetRelationships

Summary:

Returns the relationships (item and relationship type) that are defined for a given item

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns a message containing an array of relationships. Each relationship contains an item identifier and relationship type.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None.

Rationale:

Administrative tool users will need to see the relationships between items. At the very least, users will need to see the isMemberOfCollection and isMemberOfCategory relationships.

Notes:

There are no restrictions imposed by this function on the types of relationships that can be defined. The relationship types will be implementation specific and may vary repository to repository.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/rels
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response returning item relationships:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <relationships>
    <relation type="isMemberOfCollection" item="duke-002">http://{repositoryURI}/ite
    <relation type="isMemberOfCategory" item="duke-003">http://{repositoryURI}/items
  </relationships>
  <relationtypes>
    <relations type="isMemberOfCollection">http://{repositoryURI}/items/{item-id}/re
    <relations type="isMemberOfCategory">http://{repositoryURI}/items/{item-id}/rels
  </relationtypes>
</response>
```

Function: CreateRelationship

Summary:

This function creates a relationship between two items.

Parameters:

- **ItemAId** (string, required): if the relationship is expressed as a sentence, item a "is related to" item b
- **ItemBId** (string, required): if the relationship is expressed as a sentence, item a "is related to" item b
- **RelationType** (string, required): expresses the relationship between item a and item b; item a "is related to" item b

Returns:

Returns a message containing the status of the create relationship request.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **RelationshipNotSupported**: the underlying repository does not implement this type of relationship
- **ItemNotFound**: one of the specified items was not found in the repository

Side Effects:

A relationship may be created in the repository. Item in the repository may need reindexing.

Rationale:

Administrative tool users will need to create the relationships between items. At the very least, users will need to create the !isMemberOfCollection and !isMemberOfCategory relationships.

Notes:

There are no restrictions imposed by this function on the type of relationship that can be defined. The repository should define what relationship types are valid.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/rels
HTTP Method: POST
HTTP Parameters: itemid: (required) the item id to create relationship with

type: (required) the type of the relationship being created

Response:

HTTP Response Code: 201 Created
HTTP Location: http://{repositoryURI}/items/{item-id}/rels/{rel-type}/{itemb-id}

An example response returning error:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>01</responseCode>
  <responseMessage>Unable to set status, item failed validation</responseMessage>
</response>
```

Function: DeleteRelationship

Summary:

This function deletes a relationship between two items.

Parameters:

- **ItemAId** (string, required): if the relationship is expressed as a sentence, item a "is related to" item b
- **ItemBId** (string, required): if the relationship is expressed as a sentence, item a "is related to" item b
- **RelationType** (string, required): expresses the relationship between item a and item b; item a "is related to" item b

Returns:

Returns a message containing the status of the delete relationship request.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **RelationshipNotSupported**: the underlying repository does not implement this type of relationship
- **ItemNotFound**: one of the specified items was not found in the repository

Side Effects:

A relationship may be deleted from the repository. Item in the repository may need reindexing.

Rationale:

Administrative tool users will need to delete the relationships between items.

Notes:

There are no restrictions imposed by this function on the type of relationship that is being requested for deletion. If relationship exists that matches the passed parameters, that relationship is deleted

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/rels
HTTP Method: POST
HTTP Parameters: itemid: (required) the item id to delete relationship with

type: (required) the type of the relationship being deleted

Response:

An example response returning success:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>00</responseCode>
  <responseMessage>Relationship deleted successfully</responseMessage>
</response>
```

An example response returning error:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>01</responseCode>
  <responseMessage>Unable to delete relationship, relationship not found</responseMe
</response>
```

DUKE UNIVERSITY LIBRARIES

Function: GetItemInformation

Summary:

This function returns information about the item stored in the repository. The contents of that information are implementation-specific.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns a message containing information about the item stored in the repository.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None.

Rationale:

A repository may need or want to store additional information about an item that does not fit within the confines of descriptive, administrative, or structural metadata. The repository item will need some placeholder or bucket to store this information. Furthermore, it is likely that the contents of this bucket will change over time. It would be desirable for this function not to have to change as the contents of the bucket change over time.

Notes:

The function itself does not impose any restrictions on the item information that is returned by this function.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/iteminfo
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response returning item info:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<item>
  <itemInfo>
    <itemType>Image</itemType>
    <itemStatus>Complete</itemStatus>
  </itemInfo>
</item>
```

Function: SetItemInformation

Summary:

This functions sets additional information to the item in the repository.

Parameters:

- **ItemId** (type, required): the permanent repository identifier for the item being requested
- **ItemInfo** ([data], required): the item information data to be stored with the requested item

Returns:

Returns a message containing the status of the SetItemInformation request.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

Item information set on the item in the repository. Item may be flagged for indexing by the repository.

Rationale:

A repository may need or want to store additional information about an item that does not fit within the confines of descriptive, administrative, or structural metadata. The repository item will need some placeholder or bucket to store this information. Furthermore, it is likely that the contents of this bucket will change over time. It would be desirable for this function not to have to change as the contents of the bucket change over time.

Notes:

The function itself does not impose any restrictions on the item information that is passed to this function.

Presumably, the initial item creation will create the item information bucket, so that the invocation of SetItemInformation will update an existing resource rather than create a new resource.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/iteminfo
HTTP Method: PUT

A sample request setting the iteminfo:

```
PUT /items/{item-id}/iteminfo HTTP/1.1
Host {repositoryURI}
Content-Type: text/xml

<itemInfo>
  <itemType>Image</itemType>
  <itemStatus>Complete</itemStatus>
</itemInfo>
```

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <responseCode>00</responseCode>
  <responseMessage>Item Information set successfully</responseMessage>
</response>
```

Item (Descriptive Metadata)

Function: SupportedSchemas

Summary:

Returns an array of the descriptive metadata schemas supported by the item

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns a message containing all of the schemas that the item supports.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

Administrative tools may want to know what schema are supported by an item.

Notes:

Schema names are text strings with no inherent meaning, except that they need to be understood by the underlying repository.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/schemas
HTTP Method: GET

DUKE UNIVERSITY LIBRARIES

Response:

HTTP Response Code: 200 OK

An example response returning schema info:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<item>
  <schemas>
    <schema>DC</schema>
    <schema>DukeCore</schema>
  </schemas>
</item>
```

Function: GetDescriptiveMetadata

Summary:

This function returns the descriptive metadata record in the requested descriptive metadata schema

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **Schema** (string, required): the descriptive metadata schema that is being requested { DC, DukeCore?, VRACore, MODS }

Returns:

A message is returned containing the requested descriptive metadata

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **SchemaNotSupported**: the underlying repository does not implement this schema
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

Administrative tools may want to retrieve the descriptive metadata in different schemas

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/dmr
HTTP Method: GET
HTTP Parameters: schema: (optional) the requested schema for the response, no value returns DMR in its native schema

Response:

HTTP Response Code: 200 OK

An example response returning DMR:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<dmr xmlns:duke="http://library.duke.edu/metadata/dukecore" xmlns:dcterms="http://pu
  <dc:type>Advertisements</dc:type>
  <dc:title duke:role="Headline">more efficient... in miniature </dc:title>
  <dc:date duke:role="Year">1945</dc:date>
  <dc:subject duke:role="Company">Tung-Sol Electronic Tubes</dc:subject>
  <dc:subject duke:role="Product">Radio Tubes</dc:subject>
  <dc:source duke:role="Publication">Time</dc:source>
  <dc:source duke:role="Publication Type">magazine</dc:source>
  <dcterms:extent duke:role="Number of Pages">1</dcterms:extent>
  <dc:audience duke:role="Target Audience">Consumer</dc:audience>
  <dc:subject>Radio--Radio Tubes</dc:subject>
  <duke:category>radio</duke:category>
  <duke:category>1940-1945</duke:category>
  <duke:collection>adaccess</duke:collection>
</dmr>
```

DUKE UNIVERSITY LIBRARIES

Function: SetDescriptiveMetadata

Summary:

Sets the DMR of the item with the DMR that is passed as a parameter of the function.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **Schema** (string, required): the descriptive metadata schema of the DMR being set { DC, DukeCore, VRACore, MODS }
- **DMR** ([data], required): the descriptive metadata record that is to be set for the item

Returns:

This function returns the MetadataForm for the item.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **SchemaNotSupported**: the underlying repository does not implement this schema
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

This function will set the descriptive metadata section of the item to the supplied DMR. In some repository implementations, this will trigger validation of the descriptive metadata and the item. In some repository implementations, this will also trigger item for reindexing.

Rationale:

Metadata editing tools will need to edit the descriptive metadata for items. The editing tools will possibly need to communicate in schema other than that which is stored in the repository.

Notes:

In repositories where validation is supported, this function will perform validation on the submitted metadata when executed. If the repository also manages item statuses, this function will trigger updates to the item status based on the results of the metadata validation.

If metadata profiles are properly implemented in the repository, this function should be able to accept metadata in the MetadataForm schema. If metadata is submitted in the MetadataForm? schema, the metadata will be transformed into the native schema that is defined in the item's metadata profile.

At the very least, this function needs to accept metadata in its default schema and store it with the item in the repository.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/dmr
HTTP Method: PUT
HTTP Parameters: schema: (optional) the requested schema for the response, no value returns DMR in its native schema

dmr: (required) url encoded data containing the DMR

An example request (unencoded for clarity)

```
PUT /items/{item-id}/dmr HTTP/1.1
Host {repositoryURI}

schema=DukeCore;
dmr=<dmr xmlns:duke="http://library.duke.edu/metadata/dukecore" xmlns:dcterms="http:
  <dc:type>Advertisements</dc:type>
  <dc:title duke:role="Headline">more efficient... in miniature </dc:title>
  <dc:date duke:role="Year">1945</dc:date>
  <dc:subject duke:role="Company">Tung-Sol Electronic Tubes</dc:subject>
  <dc:subject duke:role="Product">Radio Tubes</dc:subject>
  <dc:source duke:role="Publication">Time</dc:source>
  <dc:source duke:role="Publication Type">magazine</dc:source>
  <dcterms:extent duke:role="Number of Pages">1</dcterms:extent>
  <dc:audience duke:role="Target Audience">Consumer</dc:audience>
  <dc:subject>Radio--Radio Tubes</dc:subject>
  <duke:category>radio</duke:category>
  <duke:category>1940-1945</duke:category>
  <duke:collection>adaccess</duke:collection>
</dmr>
```

Response:

HTTP Response Code: 200 OK

An example response returning MetadataForm:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<metadata_form>
  <errors>
    <error>
```

```
    <message>Headline is required</message>
    <field>headline</field>
  </error>
</errors>
<sections>
  <section id="item" label="Item"/>
  <section id="subjects" label="Subjects"/>
</sections>
<fields>
  <field name="type" label="Type" cardinality="single" sectionid="item" display="r
    <element name="type" type="dropdown" values="types">Advertisements</element>
  </field>
  <field name="headline" label="Title.Headline" cardinality="multiple" sectionid="
    <element name="headline" type="text" lookup="headlines">more efficient... in m
  </field>
  <field name="date" label="Date" cardinality="multiple" sectionid="item" display=
    <element name="date_role" type="dropdown" values="date_roles">Year</element>
    <element name="date" type="text">1945</element>
  </field>
  <field name="company" label="Company" cardinality="multiple" sectionid="subjects
    <element name="company" type="text" lookup="companies">Tung-Sol Electronic Tub
  </field>
  <field name="product" label="Products" cardinality="multiple" sectionid="subject
    <element name="product" type="text" values="products">Radio Tubes</element>
  </field>
  <field name="source" label="Source" cardinality="multiple" sectionid="subjects"
    <element name="source_role" type="dropdown" values="source_roles">Publication<
    <element name="source" type="text" lookup="sources">Time</element>
  </field>
  <field name="source" label="Source" cardinality="multiple" sectionid="subjects"
    <element name="source_role" type="dropdown" values="source_roles">Publication
    <element name="source" type="text" lookup="sources">magazine</element>
  </field>
  <field name="extent" label="Extent" cardinality="single" sectionid="item" displa
    <element name="extent_role" type="dropdown" values="extent_roles">Number of Pa
    <element name="extent" type="text">1</element>
  </field>
  <field name="audience" label="Audience" cardinality="single" sectionid="item" di
    <element name="audience_role" type="dropdown" values="audience_roles">Target A
    <element name="audience" type="text">Consumer</element>
  </field>
  <field name="subject" label="Subject" cardinality="multiple" sectionid="subjects
    <element name="subject" type="text" lookup="subjects">Radio--Radio Tubes</elem
  </field>
  <field name="category" label="Categories" cardinality="multiple" sectionid="cate
    <element name="category" type="text" values="categories">radio</element>
  </field>
  <field name="category" label="Categories" cardinality="multiple" sectionid="cate
    <element name="category" type="text" values="categories">1940-1945</element>
  </field>
```

```
  </fields>
</metadata_form>
```

Note - Not sure if this is the desired response. Maybe just a response message would be better, and the client would need to grab the MetadataForm? explicitly in a separate call.

Function: GetMetadataForm

Summary:

This function performs validation and transformations on either the stored or supplied DMR and produces a MetadataForm document.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **DMR** ([data], optional): if supplied, this function performs validation and transformations on the supplied DMR rather than the DMR stored in the item
- **Schema** (string, optional): must be supplied if the DMR is passed; indicates the schema of the supplied DMR, in case transformations are necessary

Returns:

This function returns a MetadataForm document

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **SchemaNotSupported**: the underlying repository does not implement this schema
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

This function will provide a metadata editing tool all of the information it needs to dynamically generate a metadata editing screen for editing an item's DMR. This allows a metadata editing tool to be developed generically, without having to write specific code in the interface which corresponds with the repository's schema.

DUKE UNIVERSITY LIBRARIES

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/metadataform
HTTP Method: GET
HTTP Parameters: schema: (optional) the schema of the DMR if it is passed

dmr: (optional) url encoded data containing the DMR

Response:

HTTP Response Code: 200 OK

An example response returning MetadataForm:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<metadata_form>
  <errors>
    <error>
      <message>Headline is required</message>
      <field>headline</field>
    </error>
  </errors>
  <sections>
    <section id="item" label="Item"/>
    <section id="subjects" label="Subjects"/>
  </sections>
  <fields>
    <field name="type" label="Type" cardinality="single" sectionid="item" display="r
      <element name="type" type="dropdown" values="types">Advertisements</element>
    </field>
    <field name="headline" label="Title.Headline" cardinality="multiple" sectionid="
      <element name="headline" type="text" lookup="headlines">more efficient... in m
    </field>
    <field name="date" label="Date" cardinality="multiple" sectionid="item" display=
      <element name="date_role" type="dropdown" values="date_roles">Year</element>
      <element name="date" type="text">1945</element>
    </field>
    <field name="company" label="Company" cardinality="multiple" sectionid="subjects
      <element name="company" type="text" lookup="companies">Tung-Sol Electronic Tub
    </field>
    <field name="product" label="Products" cardinality="multiple" sectionid="subject
```

```
      <element name="product" type="text" values="products">Radio Tubes</element>
    </field>
    <field name="source" label="Source" cardinality="multiple" sectionid="subjects"
      <element name="source_role" type="dropdown" values="source_roles">Publication<
      <element name="source" type="text" lookup="sources">Time</element>
    </field>
    <field name="source" label="Source" cardinality="multiple" sectionid="subjects"
      <element name="source_role" type="dropdown" values="source_roles">Publication
      <element name="source" type="text" lookup="sources">magazine</element>
    </field>
    <field name="extent" label="Extent" cardinality="single" sectionid="item" displa
      <element name="extent_role" type="dropdown" values="extent_roles">Number of Pa
      <element name="extent" type="text">1</element>
    </field>
    <field name="audience" label="Audience" cardinality="single" sectionid="item" di
      <element name="audience_role" type="dropdown" values="audience_roles">Target A
      <element name="audience" type="text">Consumer</element>
    </field>
    <field name="subject" label="Subject" cardinality="multiple" sectionid="subjects
      <element name="subject" type="text" lookup="subjects">Radio--Radio Tubes</elem
    </field>
    <field name="category" label="Categories" cardinality="multiple" sectionid="cate
      <element name="category" type="text" values="categories">radio</element>
    </field>
    <field name="category" label="Categories" cardinality="multiple" sectionid="cate
      <element name="category" type="text" values="categories">1940-1945</element>
    </field>
  </fields>
</metadata_form>
```

Function: GetMetadataProfileName

Summary:

This function returns the name of the metadata profile for the given item

Parameters:

- **ItemId** (type, required): the permanent repository identifier for the item being requested

Returns:

Returns a string containing the name of the metadata profile for the given item

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

If the repository supports metadata profiles, this function provides an editing tool with an identifier to retrieve the metadata profile from the repository.

Notes:

Metadata profiles are uniquely named and live inside the repository. This function provides the unique name for the metadata profile that the given item is associated with.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/metadataform
HTTP Method: GET
HTTP Parameters: schema: (optional) the schema of the DMR if it is passed

dmr: (optional) url encoded data containing the DMR

Response:

HTTP Response Code: 200 OK

An example response returning MetadataForm:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <metadata_application_profile_name>Duke Core -- Advertising</metadata_application_
</response>
```

DUKE UNIVERSITY LIBRARIES

Function: IsDescriptiveMetadataEditable

Summary:

This function returns a boolean indicating whether or not the descriptive metadata record is editable for the given item.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

This function returns a boolean value indicating whether or not the descriptive metadata record is editable for the given item

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

In a repository implementation, some items may have derived descriptive metadata, such as those with rich xml components. Some examples include TEI and EAD. For these and other similar items, an editing tool needs to be informed if it can edit the metadata.

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/dmreditable
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <editable>Y</editable>
</response>
```

Item (Component Management)

Function: GetValidComponentTypes

Summary:

This function returns a list of valid component types that can be added to the item.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns an array of valid component types.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

Editing tools will manage the components for an item. The editing tool will need to be informed what types of components can be added to an item.

Notes:

Component types are general. Some examples are Image, MovingImage?, TEI. Although they are closely related, they are not specifically MIME types.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/validcomponenttypes
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <valid_component_types>
    <type>DukeImage</type>
  </valid_component_types>
</response>
```

Function: GetComponents

Summary:

This function returns an array of component maps for an item.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns an array of component maps for an item. Each component map can include an identifier, a label, an order, a copy, a type, and possibly a relation. The identifier is the unique repository identifier for the component. The label (optional) will be a human readable label that provides some descriptive info for the component (eg. page 1, front cover, table of contents). The order indicates the order in which components are understood or displayed in interfaces. The copy indicates the copy of the component, whether it is a master copy or a display copy. The type indicates the component type (image, movingimage, etc). The relation (optional) is an isPartOf relation that indicates that the component is part of the item.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None.

Rationale:

An editing tool will need to understand the structure of an item. The component maps provide a clear map of the item's structure.

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <components>
    <component>
      <identifier>duke-003</identifier>
      <label>Front</label>
      <order>1</order>
      <copy>DISPLAY</copy>
      <type>DukeImage</type>
      <relation>isPartOf</relation>
    </component>
    <component>
      <identifier>duke-004</identifier>
      <label>Back</label>
      <order>2</order>
      <copy>DISPLAY</copy>
      <type>DukeImage</type>
      <relation>isPartOf</relation>
    </component>
  </components>
</response>
```

DUKE UNIVERSITY LIBRARIES

Function: AddComponent

Summary:

This function adds a component to an item, without adding the component's bitstream.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **ComponentMap** ([data], required): the component map for the component being added; the component map will not contain an identifier, will need to contain an order, copy and type, and will optionally contain a label and relation

Returns:

Returns a component map which contains the newly created component's identifier.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

A component is created within the repository. The item is updated with any structural metadata it needs to refer to the component. The item may be flagged/queued for indexing.

Rationale:

There may be instances where an editing tool wants to create a component prior to uploading a bitstream to it.

Notes:

The repository implementation will need to decide if it wants to implement this function. If so, some consideration will have to be paid to when administrative metadata is created for the component.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components
HTTP Method: POST
HTTP Parameters: componentmap: (required) the component map

An example request (unencoded for clarity)

```
PUT /items/{item-id}/dmr HTTP/1.1
Host {repositoryURI}

componentmap=<component>
  <identifier>duke-003</identifier>
  <label>Front</label>
  <order>1</order>
  <copy>DISPLAY</copy>
  <type>DukeImage</type>
  <relation>isPartOf</relation>
</component>
```

Response:

HTTP Response Code: 201 Created
HTTP Location: URI to the component

An example response:

```
HTTP/1.1 201 Created
Location:http://{repositoryURI}/items/{item-id}/components/{component-id}
```

Function: AddComponent

Summary:

This function adds a bitstream to an existing component or creates the component and then adds the bitstream.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **ComponentMap** ([data], required): the component map must contain enough information to either create and map a component, or uniquely identify an existing component
- **Stream** ([data], required): the bitstream to be added to the component

Returns:

Returns a component map for the component

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: in the case of specifying a component id, the component was not found in the repository

Side Effects:

A component is created in the repository, with its bitstream, and associated with an item. Presumably, all associated derivatives and behaviors are created in the repository for the added component.

Rationale:

This function allows for ease of component addition. Editing tools will need the ability to add components to the repository.

Notes:

Repositories may implement this function in different ways. At the very least, every repository will need the ability to upload components, and this function allows for that.

This function will also be implemented differently per item type. The function will create the component's derivatives and behaviors, which will be different for different item types (and different component types).

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components
HTTP Method: POST
HTTP Parameters: componentmap: (required) the component map

file: (required) the stream of data

Note - this is a multi-part form

Response:

HTTP Response Code: 201 Created
HTTP Location: URI to the component

An example response:

```
HTTP/1.1 201 Created
Location:http://{repositoryURI}/items/{item-id}/components/{component-id}
```

DUKE UNIVERSITY LIBRARIES

Function: ReplaceComponent

Summary:

This function replaces the component, which is referred to by the component map, with a new component.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **ComponentMap** ([data], required): the component map must contain enough information to uniquely identify an existing component
- **Stream** ([data], required): the new bitstream to be added to the component

Returns:

Returns a component map for the replaced component.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

Component is replaced in the repository. Derivatives are created for the new component. Administrative metadata will be cleared out.

Rationale:

There may be need to replace the bitstream and administrative metadata for a component without affecting the structure of the item. An example is replacing page 4 of a book.

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}
HTTP Method: PUT
HTTP Parameters: file: (required) the stream of data

file: (required) the stream of data

Note - this is a multi-part form

Response:

HTTP Response Code: 200 OK
HTTP Location: URI to the component

An example response:

```
HTTP/1.1 200 OK
Location:http://{repositoryURI}/items/{item-id}/components/{component-id}
```

Function: DeleteComponent

Summary:

This function deletes the component, which is referred to by the component map.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested
- **ComponentMap** ([data], required): the component map must contain enough information to uniquely identify an existing component

Returns:

Returns a message containing the status of the request.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

Component is deleted from the repository. Structural component map of the item is updated. Item will be revalidated for completeness. Item may be flagged for reindexing.

Rationale:

There may be need to completely remove a component from an item.

DUKE UNIVERSITY LIBRARIES

Notes:

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}
HTTP Method: DELETE

Response:

HTTP Response Code: 200 OK

An example response:

```
<response>
  <message>Component successfully deleted</message>
</response>
```

Item (Misc)

Function: GetThumbnailURL

Summary:

This function returns a URL for a thumbnail for the item.

Parameters:

- **ItemId** (string, required): the permanent repository identifier for the item being requested

Returns:

Returns a url

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository

Side Effects:

None

Rationale:

It might be useful to index this value so that thumbnails can be efficiently retrieved from a resultset

Notes:

The URL that is returned may be only a partial URL. This depends on whether or not the underlying repository is able to generate thumbnails at multiple resolutions. This will an implementation-specific decision though.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/thumbnailurl
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <thumbnailurl>http://...</thumbnailurl>
</response>
```

Component

Function: GetThumbnail

Summary:

This function returns a bitstream of data representing the thumbnail for the component. This will typically be an image/jpeg response.

Parameters:

- **ItemId** (string, optional): the permanent repository identifier for the item being requested, required or optional depending upon the repository implementation, specifically, whether or not component identifiers are unique and globally addressable
- **ComponentId** (string, required): the permanent repository identifier for the component being requested

Returns:

Returns an image/jpeg

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

None

Rationale:

Interfaces will need to display thumbnails of components

Notes:

As of the writing of this API documentation, it is still unclear whether or not this function will be needed in the API or whether the repository will provide this natively. Some repositories have native methods of providing this functionality.

DUKE UNIVERSITY LIBRARIES

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}/thumbnail
HTTP Method: GET

Response:

HTTP Response Code: 200 OK
HTTP Content-Type: image/jpeg

Function: GetPreviewURL

Summary:

This function returns a url to preview a component as it will be accessible to the public

Parameters:

- **ItemId** (string, optional): the permanent repository identifier for the item being requested, required or optional depending upon the repository implementation, specifically, whether or not component identifiers are unique and globally addressable
- **ComponentId** (string, required): the permanent repository identifier for the component being requested

Returns:

Returns a url

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

None

Rationale:

Users of a metadata editing tool will want to preview a component in much the same way that the public will consume the component. A metadata tool will need to allow them to do this.

Notes:

This functionality might not make sense in all repository scenarios. It may not make sense to replicate a public-facing interface within an editing tool. It may also not make sense to preview a component outside the framework of its item.

So, this function can really be implemented to return any url that makes sense to the repository and interface that might be accessing this url.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}/previewurl
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

An example response:

```
HTTP/1.1 200 OK
Content-Type: text/xml

<?xml version='1.0' encoding='UTF-8'?>
<response>
  <previewurl>http://...</previewurl>
</response>
```

Function: GetComponentMetadata

Summary:

This function returns the AMR (administrative metadata record) for a component.

Parameters:

- **ItemId** (string, optional): the permanent repository identifier for the item being requested, required or optional depending upon the repository implementation, specifically, whether or not component identifiers are unique and globally addressable
- **ComponentId** (string, required): the permanent repository identifier for the component being requested

DUKE UNIVERSITY LIBRARIES

Returns:

Returns a message containing the administrative metadata record for the requested component.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

None

Rationale:

Editing tool will need to expose the administrative metadata of the component for consumption and editing.

Notes:

There are no restrictions imposed by the function itself on the content or structure of the AMR. The repository implementation may impose restrictions or structure, but it is not up to this function to enforce either.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}/amr
HTTP Method: GET

Response:

HTTP Response Code: 200 OK

Function: SetComponentMetadata

Summary:

This function sets the AMR (administrative metadata record) for a component.

DUKE UNIVERSITY LIBRARIES

Parameters:

- **ItemId** (string, optional): the permanent repository identifier for the item being requested, required or optional depending upon the repository implementation, specifically, whether or not component identifiers are unique and globally addressable
- **ComponentId** (string, required): the permanent repository identifier for the component being requested
- **AMR** ([data], required): the administrative metadata record to set for the component

Returns:

Returns a message containing the status of the request.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **ItemNotFound**: the specified item was not found in the repository
- **ComponentNotFound**: the component was not found in the repository

Side Effects:

The component metadata (AMR) for the component is updated

Rationale:

Editing tool will need to allow editing of the administrative metadata of the component.

Notes:

There are no restrictions imposed by the function itself on the content or structure of the AMR. The repository implementation may impose restrictions or structure, but it is not up to this function to enforce either.

Binding:

Request:

URI: http://{repositoryURI}/items/{item-id}/components/{component-id}/amr
HTTP Method: PUT
HTTP Parameters: amr: (required) the url encoded xml containing the AMR

Response:

HTTP Response Code: 200 OK

MetadataApplicationProfile (MAP)

Function: GetMetadataProfile

Summary:

This function returns the complete MAP for an item (schema reference, usage guideline reference, validation and MetadataFormDefinition)

Parameters:

- **MAPName** (string, required): the permanent repository identifier for the MAP being requested

Returns:

Returns a message containing all of the parts of a MAP

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **!MAPNotFound**: the specified MAP was not found in the repository

Side Effects:

None

Rationale:

The MetadataApplicationProfile will be used by metadata tools to dynamically generate metadata editing forms. It allows for doing so without the metadata tool needing any prior knowledge of the underlying repository's use of metadata. The tool is informed of everything it needs to know from the MetadataApplicationProfile.

Notes:

Binding:

Request:

URI: http://{repositoryURI}/maps/{map-name}
HTTP Method: GET

DUKE UNIVERSITY LIBRARIES

Response:

HTTP Response Code: 200 OK
Response message contains the metadata application profile in an xml message

---

Function: GetValidationRules

Summary:

This function returns the validation rules for a MetadataApplicationProfile

Parameters:

- **MAPName** (string, required): the permanent repository identifier for the MAP being requested

Returns:

Returns the validation rules for the MAP

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **!MAPNotFound**: the specified MAP was not found in the repository

Side Effects:

None

Rationale:

Repository editing tool may want to consume the validation rules on their own. A possible reason for doing is if the repository itself does not provide validation and it is a function that the metadata editing tool must perform.

Notes:

Binding:

Request:

URI: http://{repositoryURI}/maps/{map-name}/validationrules
HTTP Method: GET

DUKE UNIVERSITY LIBRARIES

Response:

HTTP Response Code: 200 OK
Response message contains the metadata application profile's validation rules in an xml message

---

Function: GetMetadataFormDefinition

Summary:

This function returns the MetadataFormDefinition for a MAP

Parameters:

- **MAPName** (string, required): the permanent repository identifier for the MAP being requested

Returns:

Returns the MetadataFormDefinition for the MAP

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **!MAPNotFound**: the specified MAP was not found in the repository

Side Effects:

None

Rationale:

The MetadataFormDefinition will be used by metadata tools to dynamically generate metadata editing forms. It allows for doing so without the metadata tool needing any prior knowledge of the underlying repository's use of metadata. The tool is informed of everything it needs to know from the MetadataFormDefinition.

Notes:

The MetadataFormDefinition may be used on its own to generate metadata editing forms when there is not yet a DMR.

Binding:

DUKE UNIVERSITY LIBRARIES

Request:

URI: http://{repositoryURI}/maps/{map-name}/metadataformdefinition
HTTP Method: GET

Response:

HTTP Response Code: 200 OK
Response message contains the metadata application profile' metadata form definition in an xml message

ResultSet

Function: Find

Summary:

This function returns a result set of items for the given query parameters

Parameters:

- **params** (string, required): the definition of query parameters is not specified by this function, but must be understood by the index/search app

Returns:

Returns a well formed result set containing, at the very least, item identifiers.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **InvalidQueryException**: the underlying repository does not understand the request

Side Effects:

None

Rationale:

For a number of use cases, tools will need to query the repository and build result sets.

Notes:

Result sets will be made available to many interfaces through this function

Binding:

need to document here a request and response syntax

DUKE UNIVERSITY LIBRARIES

Function: Find

Summary:

This function returns a result set of items for the given query parameters in a given schema

Parameters:

- **params** (string, required): the definition of query parameters is not specified by this function, but must be understood by the index/search app
- **!returnSchema** (string, required): the schema that is desired for the result set

Returns:

Returns a well formed result set containing items in the requested schema.

Exceptional Conditions:

- **NotSupported**: the underlying repository does not implement this type of request
- **InvalidQueryException**: the underlying repository does not understand the request
- **SchemaNotSupported**: the underlying repository does not implement this schema

Side Effects:

Rationale:

For a number of use cases, tools will need to query the repository and build result sets.

Notes:

Result sets will be made available to many interfaces through this function

Binding:

need to document here a request and response syntax