EIDR 2.6 PROGRAMMERS GUIDE

Table of Contents

1 INTRODUCTION AND PREREQUISITES

This document provides an overview of the EIDR Software Development Kit's (SDK's) public APIs for the EIDR Registry and offers examples of how to use the packaged SDKs in Java and .NET (C#) programs.

The SDKs provide the recommended programmatic interfaces for EIDR. Some details are provided about the EIDR HTTP API, which is the Registry interface underpinning the Java and .NET SDKs. For details on the REST API, see the ***EIDR 2.6 REST API Reference***.

When the term API (Application Programming Interface) is used without being qualified with "HTTP", "REST", "SDK", "Java" or ".NET", it refers collectively to the various expressions of the API or operation being discussed.

This document assumes the reader is familiar with the EIDR ***Registry Technical Overview***. For details on the EIDR content data model, see the EIDR ***Data Fields Reference***. The SDK Source installation option includes additional information (as Javadoc or .NET help) that can be accessed within an IDE or as standalone documentation.

In addition, the SDKs come with a set of Command-Line Tools that perform every operation of the Registry and provide full source code examples for every API call that EIDR developers can follow. The ***Command-Line Tools Overview*** provides detailed instructions and examples using the command-line tools that can be very instructive when learning how to develop programs using the EIDR SDKs.

2 OVERVIEW AND STRUCTURE OF THE APIs

2.1 SDK OVERVIEW

The main purpose of the Java and .NET SDKs is to make it easy to build applications that use the Registry HTTP API.

The SDKs contain classes for the following purposes:

- Representing EIDR objects and requests
- Establishing and managing a connection with the EIDR Registry
- Sending requests to the Registry
- Representing and interpreting Registry responses
- Converting between object and XML representations
- Utility classes that simplify working with the classes above
- Utility classes for working with data formats not native to the Registry, such as ISAN; the EIDR bulk ingest format; and an Excel-compatible, tab-delimited output format

The principal operations that send requests to the Registry are done with the classes in org.eidr.sdk.api:

- **`AddRelationship`** (Establish a lightweight relationship between two content records.)
- **`Alias`** (Deprecate a duplicate record by redirecting it to a surviving record.)
- **`Delete`** (Alias a record to the EIDR Tombstone.)

- **`GraphTraversal`** (Retrieve parent/child records from an identified starting point.)
- **`Match`** (Retrieve information about existing records that match a set of registration data.)
- **`ModificationBase`** (Retrieve the current state of a content record to be modified.)
- **`Modify`** (Submit a content record to be modified.)
- **`Promote`** (Convert the publication Status of a record from "in development" to "valid".)
- **`Query`** (Retrieve all content records that match a query expression.)
- **`Registration`** (Submit a new content record for inclusion in the Registry.)
- **`RemoveRelationship`** (Delete a lightweight relationship between two content records.)
- **`ReplaceRelationship`** (Modify an existing lightweight relationship.)
- **`Resolution`** (Retrieve content records based on a list of one or more IDs.)
- **`StatusLookup`** (Check the current status of a system or user-defined token.)
- **`VirtualFieldsRetrieval`** (Retrieve system-generated encapsulations of a content record's free text fields.)

org.eidr.sdk.party:

- **`PartyAdmin`** (Perform Party administrative tasks, including Party creation.)
- **`PartyQuery`** (Retrieve all Party records that match a query expression.)
- **`PartyResolution`** (Retrieve Party records based on a list of one or more IDs.)

org.eidr.sdk.service:

- **`ServiceAdmin`** (Perform Video Service administrative tasks, including Video Service creation.)
- **`ServiceGraph`** (Retrieve parent/child records from an identified starting point.)
- **`ServiceQuery`** (Retrieve all Video Service records that match a query expression.)
- **`ServiceResolution`** (Retrieve Video Service records based on a list of one or more IDs.)

These classes are all derived from the `Post` class or its `UserOverride` subclass, and share several common features:[1]

- The constructor for each class takes a Boolean argument that controls debugging output. Calling constructor(true) causes the SDK to print out timestamps, HTTP header information sent to the Registry, the URL being called, the XML payload (if applicable), the XML returned from the Registry, and extra information about error conditions.
- All of the methods that send data to the Registry take an `EIDRConnection` object as an argument.
- For classes that write to the Registry,
    - There are methods to set user-defined tokens.

[1] A developer using the EIDR SDK should never call the `Post` class directly, but may call methods on the `UserOverride` subclass, which sits between `Post` and the classes that perform Registry write operations. `Post` itself has been significantly extended, including the ability to set user-defined tokens and de-dupe flags, both discussed in later sections.

- There are methods for setting the de-duplication mode.

2.2 WRITE OPERATIONS

The Registry APIs have several calls that can modify the content records of the Registry. These are Create, Modify, AddRelationship, RemoveRelationship, Delete, Alias, and Promote. For the rest of this document they are referred to as *batchable operations*.

Each of these APIs takes a single request, which in turn can be composed of multiple operations. All the operations in a request must be the same type, such as all Registration or all Modify.

The EIDR API is based on two separate concepts that sometimes be confused with each other: single/batch requests and the immediate/asynchronous (or, non-immediate) response flag.

The first thing to be aware of is that all requests to write to the EIDR Registry are really batch requests. All requests contain one or more operations, and what is often called a *single* or *non-batch* request is just a request containing one operation. A "single" request can be *immediate* or *non-immediate*. A request of more than one item (a batch) must be non-immediate.

"Immediate" and "non-immediate" refer to the point at which final results of the operations in the request are known.[2] "Immediate" means that the result of the request is available in the response to the initial request. "Non-immediate" means that the result may not be known at the time of the response to the initial request, so the client application has to check for the final result using a *request token*.

The initial request returns one or more tokens, which are used to which are used to track the status of batchable operations using the StatusLookup API. Tokens are returned even for immediate requests; although this may seem like overkill, having a single format can simplify the handling of registry responses.

In the SDK, all of the batchable operations come in multiple flavors, covering single/batch and immediate/non-immediate. For more information, see "Appendix: Summary of Batchable Calls".

2.2.1 SINGLE/BATCH

All batchable operations are submitted through the HTTP API in a single request. The SDK has separate interfaces for batches of one (single operation requests) and batches with multiple operations ("proper" batches).

All the operations in a batch must be the same (for example, all Create, all Modify, or all Delete). The Registry will return an Invalid Request Error for a batch that violates this constraint.

[2] The Request itself always returns a result synchronously.

By default, a status token is represented as a simple 19-digit numeric string.[3] One status token is returned for each operation in the batch, and one for the batch itself. If the batch contains only one operation, the batch token and the single operation token are the same. An application uses the token to query for the current status and final disposition of any non-immediate operations using `StatusLookup`.

Each item in a batch is processed separately. There is no guarantee that the operations will be processed in the same order as they were submitted in the batch. Thus, batches with operations that depend on each other (for example, creating a series and its seasons in a single batch) will have unpredictable results. However, the status tokens will be returned in order, so you can match the token to the specific operation it represents. Alternatively, you can provide user-defined tokens, introduced in the "Tokens" section.

Submission of the batch is authenticated and additional authorization checks are performed on each individual element of the batch. If an error occurs while processing an item in the batch, that item will fail; querying its status will return appropriate information. Processing of any other operations in the batch is not affected.

2.2.2 IMMEDIATE/NON-IMMEDIATE (ASYNCHRONOUS)

In order to guarantee the uniqueness of each record, EIDR subjects all record modification requests to de-duplication review if any of the object's metadata used in the de-duplication calculation have been changed. In most cases, a result is returned automatically. If there is ambiguity that cannot be resolved by the software, one of two things will happen:

- If the request is marked as immediate-response, the Registry immediately returns an error to the application, indicating the potential problem(s). In some cases, immediate-response requests return more detailed status information than non-immediate (asynchronous) requests.

- If the request is not marked as immediate-response (i.e., it is a non-immediate request), it is sent for manual de-duplication. Registry operators will review the request. Their determination is then made available via an API status request using the appropriate system-generated or user-defined token. This process is not real-time; manual review can take up to one business day.

An immediate response applies only to single requests; all multiple-request batches are non-immediate (asynchronous). If an application requests an immediate response for a batch of more than one item, the Registry returns an Invalid Request error.

2.2.2.1 WHEN TO USE IMMEDIATE AND NON-IMMEDIATE

[3] The exception being "User-Defined Tokens," described in their own section below.

Calls that require confirmation from the de-duplication system, such as registration and modification requests, are generally performed asynchronously, so the calls may not be able to provide an immediate result. All such calls return a token, which can be used to discover and track the status of the associated request. (Automated de-duplication results are generally an "approval," of the given record as not a duplicate, or a reference to a single high-probability duplicate record.)

For non-immediate results, if the de-duplication system can resolve the response without human review, the result will usually be available within a matter of seconds. In some cases, a summary of this information may be available in the initial Registry response, but it is always available via `StatusLookup` with the token. Because the de-duplication process may require manual review by the EIDR operations team, results may not be available for up to one business day.

In immediate mode, a request that would have gone into manual review either succeeds or fails immediately, providing a set of one or more existing EIDR IDs for the candidate duplicates. Requests that would not have gone into manual review succeed or fail as they would normally. The results are also available using the token returned with the initial registry response.

Most registration and modification workflows should use non-immediate requests. Immediate requests should only be used for particular situations, such as workflows that:

- Know with very high probability that the outcome will not require manual de-duplication review. For example, when registering new episodes with vetted metadata in an empty series, or when registering records for which almost any metadata differences are considered distinguishing, as with Manifestations or Edits.

- Manually or automatically review the candidate duplicates and determine that ne of the match candidates is a true match for the submitted record; that the submitted metadata should be corrected in a way that better distinguishes it; or that the request should be re-submitted in non-immediate mode for manual processing by EIDR.

The EIDR Web UI supports this last model. By default the initial request is performed immediately so the user can see the duplicate candidates and determine which of the three options outlined above to pursue. Generally, processes that use immediate mode need to treat the return of candidate duplicates as an error condition that requires manual intervention.

2.2.3 TOKENS

Any system that does not provide immediate results must provide requestors with a way of tracking the progress of a request, so the Registry returns a status token for its batchable operations: Registration, Modify, AddRelationship, RemoveRelationship, Promote, Delete, and Alias. Multi-item requests generate a token for the batch itself and one for each individual operation in the batch. These are called Request and Operation tokens, respectively:

- **Request tokens** refer to the status of the entire request body. These are returned in the */Response/RequestStatus/Token* element.[4] These are also called *batch tokens*.

- **Operation tokens** refer to individual requests, and are returned in the */Response/RequestStatusResults/OperationStatus/Token* XML element.

By default, tokens are in the form of a system-generated unique 19-digit numeric string, but user-defined tokens are also supported for both Requests and for the individual operations within a request.

Besides retrieving information related to the token from a Response, you can also retrieve information relating to a token with a StatusLookup request. Operation tokens can be used to retrieve information about the status of an individual request (for example, a single Create or Delete). Batch (Request) tokens can be used to retrieve information about the status of the batch *and* the individual items within it, as they become available. This information includes both the operation tokens and the current status of each item in the batch.

The EIDR Registry *always* generates a token for each Request and Operation.

Batches with a single item generate only a single token. This is treated as an operation token whenever information relating to it is returned from the Registry (as, for example, when it is initially generated or requested via StatusLookup).

While the individual requests within a batch are processed in an indeterminate order, the operation tokens associated with a batch are always returned in the order submitted. This allows the requestor to match the token to the associated transaction.

See the "Token Status Lookup" section and "Appendix: Token Use Examples" for more details.

2.2.4 PER-OPERATION DATA

Forced de-duplication and user-defined tokens both require the use of per-operation data. In both cases, these are provided via an array, which must be exactly the same length as the number of Operations in the Request.

2.2.4.1 USER-DEFINED TOKENS

In addition to the tokens generated by the Registry, users can provide their own tokens for requests and operations, which can be used in exactly the same way as the system-generated token the Registry returns. A user token is an identifying string, supplied by the requester, which is passed along with each request or record. This will usually consist of a representation of the submitting system's internal ID or primary key for the records being submitted.

[4] Using XPath notation to map the XML schema structure.

If given, User tokens will be echoed back in status request responses, and you can correlate API-generated tokens to your internal via the user token. If you do not supply user tokens, you will have to infer the correlation between submitted requests and the EIDR response by the order in which the system-generated operation tokens are returned within a batch.

NOTE: It is strongly advised that systems that use the API establish a mechanism for ensuring that each supplied User token is unique. Otherwise, status requests that refer to these User tokens will return results that are ambiguous.

These SDK classes support `userTokens` via the listed methods: *Registration, Modify, AddRelationship, RemoveRelationship, Promote, Delete,* and *Alias*.

- `public void setRequestUserToken(String userToken)`
- `public String getRequestUserToken();`
- `public void setOperationUserTokens(String [] userTokens);`
- `public String[] getOperationUserTokens();`

If a request has more than one operation, the length of the user tokens array must be the same as the number of operations in the request, or the SDK will generate an Error.

See the "Token Status Lookup" section and "Appendix: Token Use Examples" for more details and the source code for RegisterTool provided with the SDK for an example of how to use these calls.

2.2.4.2 FORCED DE-DUPLICATION

An application can request manual processing of de-duplication for content records with a single match above the applicable high threshold, which otherwise would be returned as duplicates by the automated review system. This applies to the following operations only in non-immediate (asynchronous) mode: *Create, AddRelationship, RemoveRelationship, Modify, Delete, Alias*, and *Promote*.

The `Registration`, `Modify`, `Alias`, `AddRelationship`, and `RemoveRelationship` classes all implement the following method:

```
public void setForcedFlags(DedupTypes [] forcedFlags);
```

forcedFlags allows the following values:

- null
- normal
- manual
- accept
- review

Setting these for immediate mode requests causes an EIDR Registry error. As with user-specified tokens, if a request has more than one operation, the length of the flags array must be the same as the number of operations in the request, or the SDK will generate an SDK Error.

The `forcedFlags` array on an SDK object starts out as `null`, meaning that no forced de-duplication occurs. Call `setForcedFlags(null)` or `setForcedFlags(normal)` after using the feature to turn it off again. Null implies a value of "normal" for the dedupMode attribute.

If you are certain that a submitted record is unique, despite its apparent similarity to the identified duplicate, and you would like to record it in the EIDR Registry, then you must either provide additional or alternate metadata sufficient to disambiguate the record and re-submit the Registry request or request a manual review by setting the force de-dupe flag to "manual".

Applications should use this feature only in very special circumstances, and users should give EIDR operations staff advance warning of its use.

NOTE: Since the force de-duplication flags are set per operation, not per batch, each individual request within a batch could have a different flag. However, we recommend that they all be the same and that you use separate batches for each different setting.

See the source code for RegisterTool, ModifyTool, and PromoteTool provided with the SDK for examples of how to use this feature.

2.2.5 RESET REQUEST TO DEFAULT STATE

This call sets the Request token, the array of operation tokens, and the array of forced flags to null:

```
public void clearOperationData();
```

2.3 READ OPERATIONS

All Registry read operations are synchronous and blocking – the Registry returns the result as quickly as it can and the calling SDK classes wait for the response before returning to the caller. The Registry read classes are:

- **GraphTraversal**
- **Match**
- **ModificationBase**
- **PartyQuery**
- **PartyResolution**
- **Query**
- **Resolution**
- **ServiceGraph**
- **ServiceQuery**
- **ServiceResolution**
- **StatusLookup**
- **VirtualFieldsRetrieval**

2.4 RETURN VALUES

The HTTP API returns a variety of XML elements, including <Response> and <AdminResponse>. Some calls return a different XML element for success and failure responses.

In the SDK, all Registry read and write requests return a Response object, defined in `org.eidr.sdk.model`, or a subclass of Response. The Response object provides a consistent interface across the whole range of Registry return values.

Each SDK call can return one of four Status Types, which are found in `org.eidr.sdk.model.StatusTypes` (for the Java SDK) or `org.eidr.schema.StatusTypeType` (for the .NET SDK):[5]

- **Success** (The request was processed by the EIDR Registry.)
- **EIDRError** (The Registry found an error in the request, such as malformed XML or an authentication error.)
- **SDKError** (The SDK was unable to convert the call into a Registry Request, for example, because of missing data or a batch that is too large for the registry to process in a single request.)
- **Error**(There was some other problem, for example, a network failure. These errors are usually generated by the Java or .NET runtime libraries.)

For more information on EIDR error codes, see "Error Types" in the ***EIDR Registry Technical Overview*** and "Codes and Descriptions" in the ***EIDR HTTP API Reference***.

A response to any Request generally behaves as follows:

On Success:

- getStatus() returns StatusTypes.Success
- getMessage() returns a human-readable response
- getResponseObj() returns the object version of Response
- getResponsePayload() returns the registry-provided XML associated with that SDK response object

On EIDRError:

- getStatus() returns StatusTypes.EIDRError
- getMessage() returns useful info extracted from the Registry response
- getResponseObj() returns the object version of the error XML
- getResponsePayload() returns the registry-provided error XML associated with that SDK response object

On SDKError:

[5] This document primarily describes the Java SDK implementation. In the .NET SDK implementation, there are some differences in the names of model constructs and how they are accessed. See "Schema Classes" for more information.

- getStatus() returns StatusTypes.SDKError
- getMessage() returns a textual description of what went wrong, e.g., "argument lengths must be equal" when two arrays are not the same length
- getResponseObj() returns null
- getResponsePayload() returns "" (the empty string, not null)

On Error:

- getStatus() returns StatusTypes.Error
- getMessage() returns status information, e.g., information from catching an internal exception
- getResponseObj() returns null
- getResponsePayload() returns "" (the empty string, not null)

Some classes extend Response to include other information. For example, resolution.resolveSimple() returns a SimpleInfoResponse, which also has a getSimpleInfo() method to call if the request was successful. All of these extended Responses are in org.eidr.sdk.model

- `AllSelfDefinedInfoResponse`
- `FullObjectInfoResponse`
- `KernelMetadataResponse`
- `PartyQueryResponse`
- `PartyResolutionResponseType`
- `ProvenanceInfoResponse`
- `ServiceQueryResponse`
- `ServiceResolutionResponseType`
- `SimpleInfoResponse`

See the source code for the tools included with the SDKs for examples of how to handle the various kinds of responses.

A successful response to a request that contains multiple operations will return at least a token for the request. StatusLookup on that token eventually will return a response that has a list of OperationStatusType objects, one for each operation in the original request, each of which will have its own unique operation token. See the source code for RegisterTool included with the SDK for an example of iterating through this list.

2.4.1 TIPS AND TRICKS

- All calls to the EIDR Registry must be via HTTPS, not HTTP
- For EIDRError, you can find the actual Registry error with Java using `responseObj.getStatus().getCode()`, `responseObj.getStatus().getType()`, and `responseObj.getStatus().getDetails()`
- For .NET, use `responseObj.getStatus().Code`, `responseObj.getStatus().Type`, and `responseObj.getStatus().Details`

- The standard errors from the Registry APIs are listed in the ***EIDR Registry Technical Overview*** and ***EIDR HTTP API Reference***.
- A `getStatus()` result of Success means only that the EIDR Registry accepted and processed the Request. The Request itself may have caused errors when the Registry tried to execute the request.

2.4.2 HTTP RETURN CODES

Generally the Registry will generate an http return code of **200** ("Success"). This means that the request made it to the HTTP API for processing and that the HTTP API could return a result.

A **400**-class error indicates that the Registry was contacted but that the request URL was incorrect. This should not happen if you are using the SDK, but may occur with applications that construct HTTP API request URLs themselves.

A **505** error indicates that the requested HTTP API function was found, but that the request could not be executed, usually because of an error in one of the parameters. This can happen because some of the information sent to the SDK API is used to construct the HTTP URL, and so must conform to the HTTP RFC. If something is provided that results in a malformed request, the Registry will return a 505 error. The most common cause of this is passing a malformed EIDR ID to the SDK. For example:

- Having leading or trailing spaces in the EIDR ID
- Mistakenly sending in a non-EIDR ID where an EIDR ID is expected

2.5 ENCODING AND ESCAPING CONVENTIONS

Special XML characters in a Registry response are escaped as appropriate, regardless of the technique (escaping or CDATA) used in the original request. The special characters and their escaped representation are:

Character	XML Escape
"	`&quot;`
'	`&apos;`
<	`<`
>	`>`
&	`&`

The parentheses characters are reserved in EIDR Query expressions. These characters need to be escaped with a backslash (e.g., \& in place of &) if used in a query.

2.6 XML-BASED AND OBJECT-BASED SDK CALLS

In the SDKs, the primary Registry operations are available in the `org.eidr.sdk.api` package or namespace. Each class contains methods that permit you to submit requests using either information objects contained within the SDK, or an XML-based request using XML strings.

For example, the `AddRelationship` class offers both `addSingleRelationshipFromObj()` and `addSingleRelationshipFromXML()` methods.

2.7 EIDR SCHEMA FILES

The version of the XML schemas that the EIDR SDKs rely upon are bundled with the SDK distribution packages. In addition, those unique to EIDR are available on the Web at http://eidr.org/schema/, with archival copies of older schemas available in appropriately named subdirectories. For example, while the 2.6 EIDR schemas are in http://eidr.org/schema/, a copy is also held at http://eidr.org/schema/2.6.0, while a copy of the (now deprecated) 1.2.1 schema is available in http://eidr.org/schema/archive/1.2.1.

Most breaking changes in the EIDR schemas only occur in "dot" releases, e.g., 2.0, 2.1, etc. EIDR "dot-dot" releases (2.1.1, 2.1.2, etc.) contain non-breaking changes, affecting Registry and de-duplication internal operations, the introduction of new features that cannot be called from older APIs, changes to elements not encapsulated in the schema (such as the text of certain error messages), and additions to existing controlled vocabularies (the latter with a forwards compatibility mode so older schema versions can reference the vocabulary entries of newer schema versions).

NOTE: The EIDR Command-LineTools, included with the SDKs, can be run from locations other than their original install directory. This means that the tools cannot reliably locate a local copy of the schemas and therefore rely upon the Web copy.

2.8 EIDR CONFIGURATION FILE

The EIDR SDK and Tools obtain their Registry credentials from a configuration file found on the local workstation. The important fields in an EIDR configuration file are:

- <user> This is your user ID, provided by EIDR operations. It looks something like 10.5238/your-name. The prefix will always be 10.5238
- <party> This is the party name for you user ID. All users are associated with a party. It is of the form 10.5237/9DD9-E249. The prefix is always 10.5237.
- <url> This is the URL for the registry you are using. The valid choices are
    - https://registry1.eidr.org:443/EIDR for the production registry.
    - https://resolve.eidr.org:443/EIDR for the production read-only mirror.
    - https://sandbox1.eidr.org:443/EIDR for the test registry.

Sometimes a special-purpose registry may be used for pre-release testing, in which case alpha and beta users will be given the appropriate URL.

- <PageSize> Query results are returned in sets or "pages", and this gives the maximum number of results to return per page. Larger values of PageSize are generally more efficient since they amortize the cost of the HTTP request over more results. Some of the tools internally use a small PageSize if all they are interested in is the number of results rather than the results themselves.

If you are connecting to the EIDR Registry via a proxy server, then additional configuration parameters may be necessary:

- <proxyhost> This is the hostname of the proxy. This should be either an IP address or a domain name that is resolvable by the name service running on your machine.
- <proxyport> This is the port number that the proxy is attached to on the <proxyhost>.
- <proxyauthuser> The user sent for proxy authentication. Do not use if there is no authentication required.
- <proxyauthpass> The password sent for proxy authentication. Do not use if there is no authentication required.

NOTE: The Java SDK proxy uses the underlying proxy support in the JVM, and as such, any limitations in the base JVM proxy support will limit the SDK proxy support.

The Tools and SDK come with a sample configuration file: Doc/Examples/eidr-config-sample.xml

Add your EIDR credentials (partyname, username, and password, obtained from EIDR Operations) to the sample configuration file and place the file in your home directory as `eidr-config.xml`

- Unix: $HOME/eidr-config.xml
- Windows 7, 8, 10: \Users\[windows-username]\eidr-config.xml
- Mac: /Users/[mac-username]/eidr-config.xml

Tools that use API calls that do not require credentials (generally limited to Resolve) do not require credentials. For those tools credentials can be left out, but if present they must be valid.

3 SUPPORTING CLASSES IN THE SDK

3.1 CONNECTION CLASSES

The `EIDRConnection` class manages information needed to construct HTTP calls from the SDK, including the identification and authentication information necessary for a successful Registry request.

3.2 RESPONSE CLASSES

The SDK response classes are defined in `org.eidr.sdk.model`, which contains the `Response` parent class. The following classes are derived from `org.eidr.sdk.model.Response`:

- Responses to Resolution requests
  - `AllSelfDefinedInfoResponse`
  - `FullObjectInfoResponse`
  - `KernelMetadataResponse`
  - `ProvenanceInfoResponse`
  - `SimpleInfoResponse`
- Responses to Party requests
  - `PartyQueryResponse`
  - `PartyResolutionResponseType`
- Responses to Video Service Requests
  - `ServiceQueryResponse`
  - `ServiceResolutionResponseType`

3.3 SDK UTILITY CLASSES

3.3.1 SCHEMA CLASSES

3.3.1.1 JAVA

For the Java SDK, the schema classes are:

- Classes and constants generated from the DOI schemas:
  - org.doi._2010.doischema
  - org.doi._2010.iso3166a2
- Classes and constants from the MovieLabs Common Metadata schema:
  - com.movielabs.schema.md.v2_1.md
- Classes and constants derived from the EIDR schemas:

- org.eidr.schema

For example, the XML complex type `requestType` is defined as `RequestType` in `org.eidr.schema.RequestType.java`.[6]

3.3.1.2 .NET

In general, object names and methods in .NET mirror their corresponding Java SDK entities, except for the use of Properties in the place of java's getter and setter methods, and the use of object arrays to store some child objects.

All the schema classes are in the `org.eidr.schema` namespace defined in `EidrSchema.cs`. For example, the XML complex type `requestType` mentioned above is defined as the class `responseType` in the class `org.eidr.schema.requestType` in `EidrSchema.cs`.

.NET model objects derived from the XML schema may have slight capitalization variations or appended qualifiers that Java does not. For example, the schema's `eidr:statusType` enumeration is named `StatusTypes` in Java, but `statusTypeType` in .NET.

3.3.2 FIELD ACCESS

3.3.2.1 JAVA

Use the JAXB bindings to access the fields. In general, each XML type is represented as a class, with one or two methods representing each element and attribute:

- For fields with cardinality 0 or 1, use the getter to read the field and the setter to write the field.
- For fields that can have multiple instances, the single function returns a live List<> object containing the values. Use the List<>'s get() method to read values which can then be modified. It is also possible to use the add(), clear(), remove(), etc. methods on the List<> for more complex modifications.

NOTE: It is important to check for null values and List<> objects whose size() method returns 0. See the source code for FlattenTool included with the SDK. It reads every possible field in the registry.

3.3.2.2 .NET

In the .NET model implementation, an object's children are represented as an object array wherever the schema defines an `xs:choice` of one or more `xs:sequence`. In these instances, the model-

[6] Technically, path names are represented using subdirectory notation (org/eidr/schema/ResponseType.java), while class names are represented using dot notation (org.eidr.schema.ResponseType.java). For visual consistency, we use dot notation and leave it to the reader to switch to subdirectory notation when necessary.

equivalent parent object has an `Items` member (array) that contains a sequence of child objects, and a corresponding `ItemsElementName` member. `ItemsElementName` is an array that contains a list of enumerated type elements declaring the type of object that is in the `Items` array at the same index.

For example, an XML `<Response>` tag may contain `<RequestStatus>` and `<RequestStatusResults>` elements. The .NET model instance of `responseType.Items` will be an object array containing `[requestStatusType, requestStatusResultsType]` instances and a `responseType.ItemElementName ResponseChoiceType` array containing `[ResponseChoiceType.RequestStatus, ResponseChoiceType.RequestStatusResults]` enumerations.

To assist in extracting children from models objects following this implementation pattern, there are two utility classes: `org.eidr.sdk.utils.ObjResponseUtils` and `org.eidr.sdk.utils.ObjMetadataUtils`.

For example, to extract the `RequestStatus` element from `ResponseType`, simply use:
`ObjResponseUtils.GetFirstRequestStatus(responseObj);`

The "First" in the method name above refers to the fact that only the first `requestStatusType` object in the object array will be returned. Though the schema only allows one `RequestStatus` child element of `ResponseType`, the model implementation allows several. The "First" distinction is made because there are some objects where multiple instances are indeed permitted by the schema.

For example,
`ObjResponseUtils.GetFirstRequestStatusResultsAllOperationStatus(responseObj)` returns an array of `operationStatusType` children of the `RequestStatusResults` member of a `<Response>` object.

If fields are added to or removed from these collections, the Items and ItemsElementName arrays themselves should be reconstructed to have the correct values and count.

3.3.3 SERIALIZATION CLASSES

3.3.3.1 EIDRXMLOBJUTILS CLASS (JAVA)

The Java SDK contains an `org.eidr.sdk.utils.EidrXMLObjUtils` class that provides methods for serializing objects into XML. This class is primarily a helper function for the SDK, but can be a used to convert various SDK classes to XML and back again. This is not meant to be a generic object for XML conversion.

To convert XML to an object, use:

```
public static Object XMLAndTypeToObj( String xml, Class<?>
expectedType, int classtype );
```

To convert an object to XML use:

```
public static String objAndTagToXML(java.lang.Object obj, String
tag, String nsUri, String nsPrefix, int classtype)
```

`tag` is the name of the root XML Element for the object. Namespace strings are defined in `org.eidr.sdk.conf.Namespaces`. The namespace prefix is generally "" (the empty string), since the EIDR namespace is unqualified. For example, if the object is SimpleInfoType (the return type for queries):

```
String xml = EidrXMLObjUtils.objAndTagToXML(simple,
"SimpleMetadata", Namespaces.EIDR_URI, "",
EidrXMLObjUtils.EIDR_Obj_SimpleInfoType);
```

3.3.3.2 SERIALIZEUTILS CLASS (.NET)

The .NET SDK contains an `org.eidr.sdk.utils.EidrXMLObjUtils` class that provides methods for serializing objects into XML similar to the java implementation.

3.3.4 CONVERTERS

3.3.4.1 BASEINFOCONVERTER CLASS

The `org.eidr.sdk.utils.BaseInfoConverter` class contains methods that convert between `CreationFullInfo` and `CreationSelfDefined` objects, in addition to utility methods for cloning an `AlternateIDType` and `PersonNameType`.

3.3.5 FORMATTERS

3.3.5.1 XMLFORMATTER CLASS

The `org.eidr.sdk.utils.xml.XmlFormatter` class contains methods that convert raw XML into a human-readable format.

3.3.6 TYPE UTILITIES

The `org.eidr.sdk.utils.TypeUtils` class contains methods that determine if instances of `FullObjectInfoType` or `SimpleInfoType` are root objects, or if an Object has a particular relationship. See the source code for FlattenTool included with the SDK for examples.

4 CONTENT RECORD APIs

4.1 READ

4.1.1 RESOLUTION

Resolution provides various views from the Registry of the metadata associated with a particular EIDR ID.

4.1.1.1 USAGE

EIDR provides methods for performing the following types of resolution:

- **Full** (All content metadata associated with a particular record, whether self-defined or inherited.)
- **Self-defined** (All content metadata directly recorded in a particular record, excluding any inherited values.)
- **Simple** (A high-level summary view of the Full metadata record.)
- **Provenance** (A summary of a record's editorial history, including its creation, last modification, and version number.)
- **DOI Kernel** (A generic view of a record's metadata specified by the International DOI Foundation.[7])

The `org.eidr.sdk.api.Resolution` class contains the following methods:

```
public KernelMetadataResponse resolveDOIKernel(EIDRConnection
eidrConn, String assetID, boolean followAlias)

public FullObjectInfoResponse resolveFull(EIDRConnection eidrConn,
String assetID, boolean followAlias)

public ProvenanceInfoResponse resolveProvenance(EIDRConnection
eidrConn, String assetID, boolean followAlias)

public AllSelfDefinedInfoResponse resolveSelfDefined(EIDRConnection
eidrConn, String assetID, boolean followAlias)

public SimpleInfoResponse resolveSimple(EIDRConnection eidrConn,
String assetID, boolean followAlias)
```

The different Response types provide convenience methods for retrieving the actual metadata.

For example:

[7] See *Appendix 6 DOI® Kernel Metadata Declaration: XML Schema* at www.doi.org/handbook_2000/appendix_6.pdf.

```
SimpleInfoResponse simple = res.resolveSimple(cfg.getConnection(),
id, true);
String returnedID = simple.getSimpleInfoType().getID().getValue());
```

4.1.1.2 ALIASED RECORDS

When a duplicate record is found within the Registry, it is deprecated (aliased) so that its ID points to the correct record. This ensures that there is only ever one active EIDR record for any given object. The aliased ID itself is never deleted, but the Registry's response to a read operation on that ID depends on how the followAlias flag has been set:

- If `followAlias` is `true` (the default), information returned is from the Registry record to which the Alias refers, not from the aliased record itself.
- If `followAlias` is `false` and the Registry record is not aliased, the information likewise comes from the record itself.
- If `followAlias` is `false` and the Registry record is aliased, an `AliasContinuation` is returned.

Most applications will set `followAlias` to `true`. See "Aliases and Deletion Model" in the ***EIDR Registry Technical Overview*** for more information.

NOTE: In rare circumstances, an EIDR record may be deleted rather than aliased. For example, if a test data record is accidentally created in the production Registry. In these cases, the EIDR ID still exists, but is aliased to the EIDR Tombstone record – a special record that exists solely for this purpose. The EIDR Tombstone record has EIDR ID 10.5240/ 0000-0000-0000-0000-0000-X.

For each resolutionType, the defined values and behavior are as follows:

ResolutionType	Follow Alias?	Return Value
`DOIKernel`	`true`	Returns metadata formatted as a `doi:kernelMetadata` record. If the alias chain is too deep, returns an `AliasContinuation` object. For an `InDev` object, returns a structural type of restricted. The object's name is "`No information available`". For a deleted object, returns a structural type of "Restricted", and field values from the standard tombstone object.

ResolutionType	Follow Alias?	Return Value
`DOIKernel`	`false`	If the object is aliased this returns an `doi:kernelMetadata` record where: • structural type is "Restricted" • title is "aliased" • `referentCreation.identifier.type` is DOI • `referentCreation.identifier.value` is the DOI to which the object is aliased. For a deleted object, returns a structural type of "Restricted", and field values from the standard tombstone object.
`Simple`	`true`	Returns metadata formatted as `eidr:simpleInfoType`. This is the format returned by queries and traversals. If the alias chain is too deep, returns an `AliasContinuation` object.
`Simple`	`false`	If the object is not aliased, returns an `eidr:simpleInfoType` record. If the object is aliased, returns an `AliasContinuation` object. (See above.)
`Full`	`true`	Returns metadata formatted as `eidr:fullObjectInfoType`. If the alias chain is too deep, returns an `AliasContinuation` object.
`Full`	`false`	If the object is not deleted or aliased, returns an `eidr:fullObjectInfoType` record. If the object is aliased, returns an `AliasContinuation` object. (See above.)
`SelfDefined`	`true`	Returns metadata formatted as an `eidr:allSelfDefinedInfoType` record. If the alias chain is too deep (beyond five levels), returns an `AliasContinuation` object. (See above.)
`SelfDefined`	`false`	If the object is not deleted or aliased, returns an `eidr:allSelfDefinedInfoType` record. If the object is aliased, returns an `AliasContinuation` object. (See above.)
`Inherited`	`true`	Returns metadata formatted as `eidr:allInheritedInfoType`. If the alias chain is too deep (beyond five levels), returns an `AliasContinuation` object. (See above.)

ResolutionType	Follow Alias?	Return Value
`Inherited`	`false`	If the object is not deleted or aliased, returns an `eidr:allInheritedInfoType` record. If the object is aliased, returns an `AliasContinuation` object. (See above.)
`Provenance`	`true`	The provenance record for object (with the alias chain followed) as an `eidr:provenanceInfoType` record. If the alias chain is more than too deep, returns an `AliasContinuation` object.
`Provenance`	`false`	The provenance record for the object itself as an `eidr:provenanceInfoType` record. For details on this resolution see the ***EIDR Data Fields Reference***.

4.1.1.3 TIPS AND TRICKS

4.1.1.3.1 INHERITANCE

Most inheritable fields use simple inheritance – the value is taken directly from the parent, which in turn may have taken it from its parent, and so on up the inheritance chain until an object is found that defines the field. In these cases, a field's value is one of either the self-defined metadata or the inherited metadata. Full metadata requests return all data associated with a record (both inherited and self-defined), while self-defined metadata requests exclude in inherited and return only the self-defined metadata (as one would expect).

4.1.1.3.2 PROVENANCE

Provenance contains information about the user account that created and most recently modified a record. This information is not included in the Provenance resolution if the requester is not associated with the Party that registered the record. In that case, only the dates and version number are returned.

4.1.1.3.3 DETECTING AN ALIAS

An alias can be detected as follows:

- When `followAlias` is true, by comparing the ID field of the returned object with the input ID; if they differ, the input ID is aliased.
- When `followAlias` is false, by detecting that an `AliasContinuation` has been returned rather than formatted metadata.
- Aliases are followed for five levels. If the final item is still aliased, an `AliasContinuation` object is returned that contains the last aliased ID found and the item to which it is aliased.

4.1.2 VIRTUAL FIELDS RETRIEVAL

Virtual fields are text fields constructed by combining some of the text-based metadata fields. Virtual fields are used for debugging and for certain kinds of shortcuts in searches, for example by looking for content records where a person's name appears in any of the Resource Name, Alternate Resource Name, Director, or Actor fields. Because they are virtual, and are not actually present in the EIDR records, they may be queried and retrieved, but not updated directly.

An object has two virtual string fields that combine multiple other fields. One uses full metadata from the object, including inherited fields, and one uses only the fields defined on the object itself (self-defined). Both of these are normalized (punctuation stripped, spaces collapsed) and tokenized using the default (English) rules, as are the queries run against them. Stop-word filtering and stemming are not applied.[8] This also applies to strings as queries on the virtual fields. Note that many of the components of the virtual field are names and titles, which are not filtered or stemmed on their own, and this rule applies to the whole virtual field.

- Full – composed of these fields from FullObjectMetadataType:
  - `BaseObjectData/ResourceName`
  - `BaseObjectData/AlternateResourceName` (using all that are present)
  - `BaseObjectData/Description`
  - `BaseObjectData/Credits/Director/DisplayName` (using all that are present)
  - `BaseObjectData/Credits/Actor/DisplayName` (using all that are present)
- Self-defined – same as Full, but using only those fields that have not been inherited.

In queries, the virtual fields are:

```
/VirtualField/Full
/VirtualField/SelfDefined
```

The order of the tokens within each of the constituent fields is preserved, but the order in which the constituent fields are added to the virtual field is not defined. This means that an exact match query (using `<field> "<string">`) returns unpredictable results when the query contains tokens that appear in more than one field. Virtual fields match token sequences within individual fields predictably as it would on a single field.

The SDK provides the `org.eidr.sdk.api.VirtualFieldsRetrieval` class with the following methods:

```
public Response getVirtualFields( EIDRConnection eidrConn, String
assetID)
```

[8] See "Appendix B: Text Processing and Queries" in the *EIDR Registry Technical Overview* for more information.

Once you have a successful Response, in Java the two virtual fields are retrieved with:

```
Response resp;
...
String full = resp.getResponseObj().getVirtualFields().getFull();
String self =
resp.getResponseObj().getVirtualFields().getSelfDefind();
```

In .NET, they are retrieved with:

```
Response resp
...
String full = ObjResponseUtils.GetVirtualFields(resp).Full;
String self = ObjResponseUtils.GetVirtualFields(resp).SelfDefined;
```

4.1.3 QUERY

The Query API is the primary mechanism for searching the EIDR content record database.[9] Queries are built using text-based search expressions that describe matching criteria for metadata fields. These are documented in the ***EIDR Registry Technical Overview***. The simplest query tests a single metadata field. More complex queries can be built up by grouping simple queries together with standard logical expressions.

There are two kinds of queries:

- General queries return all content records that meet the query criteria.
- Rooted queries return only content records that meet the query criteria and are descended from a specified EIDR ID.

`org.eidr.sdk.api.Query` provides a single method:

```
public Response generalQuery(EIDRConnection eidrConn, QueryType request)
```

Since there can be many results, the Response mechanism is paged. All the information needed for a query is provided to the API in the `QueryType` structure:

- **ID** (The ID for a rooted query, null otherwise.)
- **Expression** (The query string. For details, see the ***EIDR Registry Technical Overview***.)
- **Page Size** (The number of records to return per Response.)
- **Page Number** (The Page for which to return results. Numbering starts with 1. 0 indicates that all results should be returned.)
- **Continuation Token** (This should be set to null on the first request. For subsequent requests, it should be set to the value returned by the Registry in the previous Response.)

[9] Excluding EIDR ID or token resolution, of course.

If the response status is Success, the Response object contains a `QueryType`, which is the original query, and a `QueryResultsType`, which contains any results.

QueryResultsType contains:

- **`CurrentSize`** (The number of records in this chunk. Equal to `pageSize`, except perhaps for the last page.)
- **`TotalMatches`** (The total number of records, across all pages, that can be returned.)
- **`ContinuationToken`** (See above. Returned as null on the last page.)
- A list of `SimpleInfoType` records for all objects that match the query. This list will have `CurrentSize` elements.

In addition to the standard errors, this API can return `BadQuery` and `ResultTooLong`.

4.1.3.1 JAVA EXAMPLE

```
Query qryi = new Query( debugTraceState() );
Response queryResponse = null;
queryResponse = qryi.generalQuery(cfg.getConnection(), req);

if (queryResponse.getStatus() != StatusTypes.Success) {
       // handle errors
}
else {
       QueryType retQuery = queryResponse.getResponseObj().getQuery();
       QueryResultsType queryResults =
queryResponse.getResponseObj().getQueryResults();
       ...
}
```

4.1.3.2 .NET EXAMPLE

```
Query qryi = new Query(debugTraceState());
Response queryResponse = null;
queryResponse = qryi.generalQuery(cfg.getConnection(), req);

if (queryResponse.getStatus() != StatusTypes.Success)
{
  // handle errors
  }
  else
  {
    queryResponse = qryi.generalQuery(cfg.getConnection(), req);
    queryType retQuery = ObjResponseUtils.GetFirstQuery(queryResponse);
    queryResultsType queryResults = ObjResponseUtils.GetFirstQueryResults(q
  ueryResponse);
    ...

  }
}
```

4.1.3.2.1 TIPS AND TRICKS

See the QueryTool source code provided with the SDK for examples of iterating through multiple pages of results and how to build the `QueryType` object. This operation is made simpler by using

the `QueryIterator` that is defined in org.eidr.tools.QueryBaseTool. (The source code for all command-line tools is included with the SDK.) The iterator will return to the caller the query results IDs one at a time, taking care of multiple page results. So, instead of writing code to submit a request for each page of results, you can write code in the following form to go through all the results:

```
QueryType qtype = new QueryType();
.
.
Query qryi = new Query( false );
QueryIterator qi = new QueryIterator( cfg qryi, qtype );
while( qi.hasNext() ) {
    String id = qi.next();
.
.

}
```

A status of Success does not mean that there were any content records that matched the query, only that the query was properly processed. If there were matches, then Total Results will be greater than 0.

For the most efficient use of connection and transport resources, use a large Page Size, especially if you do not need to display the results interactively.

By default, the results are sorted by the Registry in strict Unicode order (essentially, alphabetically) based on the Resource Name (primary title).

4.1.4 GRAPH TRAVERSAL

EIDR content records may be connected to other content records by inheritance relationships, dependence relationships, and lightweight relationships.[10] While it is possible to traverse these networks using queries or a series of resolutions, the Graph Traversal API provides a specialized mechanism for exploring the content through its relationship structure.

The `org.eidr.sdk.api.GraphTraversal` class provides methods for all of the supported graph traversals. In general:

- Each traversal takes an ID to use as its starting point.
- For traversals that accept filter criteria, each type of filter (ReferentType, StructralType, etc) can contain 0 or more items. The number of items for each type is limited by the number of

[10] See "Relationships" in the *EIDR Registry Technical Overview* for more information.

values for that type. For example, a StructuralType filter can only have at most four values in the list, since there are only four structural types.

- Many of the traversals provide an indication of how many generations above or below each result is from the starting ID.
- All the functions except GetSeriesAncestry return items with the metadata formatted as `SimpleInfoType` in the SDKs and as a <SimpleMetadata> XML element in the HTTP API.
- Generations above and generations below are represented in the Response.

Command	**Return Values For**			**Filters**
	A Root Node	**An Internal Node**	**A Leaf Node**	
FindAncestors	0 items	1 or more items, with generations above	1 or more items, with generations above	ReferentType RelationshipType StructuralType
GetRemotestAncestor	self, GenerationsAbove =0	1 item, with generations above	1 item, with generations above	
FindDescendants	1 or more items, with generations below	1 or more items, with generations below	0 items	ReferentType RelationshipType StructuralType
GetLeafDescendants	1 or more items, with generations below	1 or more items, with generations below	self, generationsBelow= 0	
GetLightweight Relationships	0 or more items	0 or more items	0 or more items	
GetDependents	0 or more items	0 or more items	0 or more items	
GetChildren	1 or more items	1 or more items	“no children” error	
GetParent	“no parent” error	1 item	1 item	

Command	Return Values For			Filters
	A Root Node	**An Internal Node**	**A Leaf Node**	
GetSeriesAncestry	Series: <Origin> is the Series itself; no other nodes	Season: <Origin> is the Season; <SeriesBasicData> is the Series Other internal node (e.g., Edit with children): <Origin> is the object itself	Episode: <Origin> is the episodes; <SeriesBasicData> and (if these is a Season) <SeasonBasicData> are its ancestors Child of an Episode (e.g. an Edit or Manifestation): <Origin> is the item itself; <SeriesBasicData>, <SeasonBasicData>, and <EpisodeBasicData> are filled in Other leaf node: <Origin> is the item itself	

The three fundamental methods in `org.eidr.sdk.api.GraphTraversal` are:

```
public Response findAncestors(EIDRConnection eidrConn,
FindAncestorsType request)
```

This method returns `SimpleInfoType` records and `GenerationsAbove` for all ancestors which match the filters in the request. Empty lists match anything. Only inheritance relationships are considered (for example, none of the 'lightweight relationships' count). The traversal stops when it encounters an object with no ancestors.

`generationsAbove` is 1 for a parent, 2 for the parent's parent, etc.

The object itself is not returned.

```
public Response findDescendants(EIDRConnection eidrConn,
FindDescendantsType request)
```

This method returns `SimpleInfoType` records and `GenerationsBelow` for all descendants that match the filters in the request. Empty lists match anything. Only inheritance relationships are considered (for example, none of the 'lightweight relationships' count). The traversal stops when it encounters an object with no descendants.

`generationsBelow` is 1 for a child, 2 for a child's child, etc.

The object itself is not returned.

```
public Response getDependants(EIDRConnection eidrConn,
GetDependantsType request)
```

This method returns a list of `SimpleInfoType` records that depend on the input ID but that are not part of an inheritance relationship or a lightweight relationship with it.

This can be used to find composites, compilations, and manifestations that reference a particular ID.

NOTE: The remaining traversal functions are provided as a convenience; they could all be implemented by parsing the results of queries, resolutions, and the primary traversals listed above.

```
public Response getSeriesAncestry(EIDRConnection eidrConn,
GetSeriesAncestryType request)
```

This method returns a Response with a `SeriesAncestryType` result that contains:

- `SimpleInfoType` result for the input ID.
- `SimpleInfoType` and `EpisodeInfoType` result for the nearest ancestor (which can be the input ID) that has episode data. Not present if there is no such ancestor.
- `SimpleInfoType` and `SeasonInfoType` result for the nearest ancestor (which can be the input ID) that has season data. Not present if there is no such ancestor.
- `SimpleInfoType` and `SeriesInfoType` result for the nearest ancestor (which can be the input ID) that has series data. Not present if there is no such ancestor.

The Registry validation rules guarantee that there can be no more than one ancestor of each of the types.

```
public Response getLightweightRelationships(EIDRConnection
eidrConn, GetLightweightRelationshipsType request)
```

This method returns `SimpleInfoType` result for each content record to which the input ID has a non-inheriting (i.e., Lightweight) relationship.

```
public Response getRemotestAncestor(EIDRConnection eidrConn,
GetRemotestAncestorType request
```

This method returns a `SimpleInfoType` result and `GenerationsAbove` for the most distant ancestor (the root node of the identified registration tree).

If the input object has no ancestors (it is a root record), the input object is returned.

If the object has ancestors, this is the equivalent of looking for the highest-numbered `GenerationsAbove` in the return values from `FindAncestors` with no filters.

```
public Response getLeafDescendants(EIDRConnection eidrConn,
GetLeafDescendantsType request)
```

This method returns a `SimpleInfoType` result and `GenerationsBelow` for all descendants that have no descendants of their own (the leaf nodes of the identified registration tree). Note that `GenerationsBelow` may not be the same for all leaf nodes.

If the input object has no descendants, the input object itself is returned.

If the object has descendants this can also be done by interpreting the results from `GetDescendants` with no filters, but that is a little tricky since not all leaf descendants will be at the same level; you really do have to build a tree from the returned values to achieve the same result.

```
public Response getParent(EIDRConnection eidrConn,
GetParentType request)
```

This method returns a `SimpleInfoType` result for the input ID's parent, and returns an error of `NoParent` if the input object is already at the top of a tree. This is useful if all you have is an ID, but not any of its metadata, and want to know its parent. It can also be done by looking for `GenerationsAbove`=1 in the results of `GetAncestors`.

```
public Response getChildren(EIDRConnection eidrConn,
GetChildrenType request)
```

This method returns `SimpleInfoType` result for all of the input ID's immediate descendants, and returns an error of `NoChildren` if the input ID is a leaf of a registration tree. Alternatively, this can be implemented by looking for `GenerationsBelow`=1 in the results of `GetAncestors`.

4.1.4.1 TIPS AND TRICKS

Remember that a content record can only inherit directly from one parent.

The variety of return types for these calls can be daunting. See the source code for FlattenTool and GraphTool provided with the SDK for examples.

4.1.5 MATCH

It is possible to obtain de-duplication results without submitting a Create or Modify request to the Registry by using the Match API.

This API call is similar to the `Register` API, but the response is more like a query. If there are no errors, then the response (Operation Status) will show success. If there are no entries in the

duplicate results list, then no existing records were found that scored above the low threshold and the submitted record would be accepted as a new registration. If there are one or more entries in the duplicate results list, then a number of potential duplicates were found. Each item in the duplicate list includes matching scores to indicate what would happen at registration time (records with duplicate candidates that score above the low threshold go into manual de-duplication while a records with a single duplicate candidate that is above the high threshold would be returned as a duplicate registration).

The `org.eidr.sdk.api.Match` class has two methods:

```
public Response matchSingleFromObject(EIDRConnection eidrConn,
Object req )
```

```
public Response matchSingleFromXML(EIDRConnection eidrConn,
String bareXML )
```

The arguments for these calls are analogous to the equivalent methods in the Registration class, and return information about EIDR content records that match the arguments.

4.2 WRITE

This section covers the batchable write operations.

See the following sections in the ***EIDR Registry Technical Overview*** for some essential underlying concepts:

- Content Records
- Categorization of Objects
- Content Record Creation and Modification

On success, all the calls in this section return a response with a Request token, which can be used to discover individual operation tokens if the request contained more than one item. The order in which errors are generated is:

- **Authentication Errors** (No further processing is done if the submitted credentials are invalid.)
- **XML Validation Errors** (No further processing is done if the submitted XML does not pass schema validation – e.g. is missing a required field or has an illegal field value.)
- **Authorization Errors** (This is performed separately for each item in the batch.)
- **Rule-based Validation Errors** (These are enforced by the Registry's data validation rules, and are evaluated separately for each item in a batch. The rules are covered in an Appendix to the ***EIDR Data Fields Reference***.)
- **Operation-specific Errors** (The specific errors and responses are detailed below.)

4.2.1 REGISTRATION

The `org.eidr.sdk.api.Registration` class has the following methods:

```
registerSingleFromXML(EIDRConnection conn, String bareXML,
boolean bImmediate)

registerBatchFromXML(EIDRConnection conn, String[] xml)
```

The XML passed to these functions must of the form:

```
<Create type="CreateBasic">
      <Basic>...</Basic>
</Create>
```

or

```
<Create type="CreateSeries">
      <Series>...</Series>
</Create>
```

The XML generated by objects such as `CreateBasicDataType` and `CreateSeriesDataType` is of the form:

`<Basic>...</Basic>` or `<Series>...</Series>`.

See below for helper functions in the Registration class to add the `<Create type="…">` boilerplate around the XML.

```
Registration.registerSingleFromObject(EIDRConnection conn,
Object obj, boolean bImmediate)

Registration.registerBatchFromObj(EIDRConnection conn, Object
obj[])
```

The object must be of a type such as `CreateBasicDataType` or `CreateSeriesDataType`. Otherwise these calls will return an `SDKError`.

It is common to get a `ValidationError` for registration requests. If the error details indicate invalid XML, this either means that a required field is missing in the created type or that an illegal value has been used for a field. Other validation errors relate to the constraints enforced by the Registry's validation rules.

4.2.2 MODIFY

Modifying objects is done by retrieving the current version of an object from the Registry, modifying it, and re-submitting it to the Registry. Use a `GetModificationBase` request to retrieve the information, and a `Modify` request to submit it once it has been changed. See "Appendix: Notes on Modifying Records" for information on how to use `GetModificationBase`.

A modify operation cannot change the presence or type of a relationship, so it is not possible to change a root record into a child record or a child record into a root record using a Modify request. Nor is it possible to change the Registrant or the Status fields.

The `org.eidr.sdk.api.Modify` class has the following methods:

```
modifySingleFromXML(EIDRConnection conn, String bareXML,
boolean bImmediate)

modifyBatchFromXML(EIDRConnection conn, string[] xml)
```

The XML passed to these functions must of the form

```
<Modify type="CreateBasic">
      <ID>10.4240/....</ID>
      <Basic>...</Basic>
</Modify>
```

or

```
<Modify type="CreateSeries">
      <ID>10.4240/....</ID>
      <Series>...</Series>
</Modify>
```

The value contained in the Response for `GetModificationBase(…,CreationType.CREATE_BASIC)` is a `CreateBasicDataType`, and the XML for that object is `<Basic>...</Basic>`. See "Helper Functions for Register and Modify" for utilities to add the necessary headers.

```
Modify.modifySingleFromObject(EIDRConnection conn, Object obj,
String id, boolean bImmediate)

modify.modifyBatchFromObj(EIDRConnection conn, Object obj[],
String id[])
```

The object must be of a type such as `CreateBasicDataType` or `CreateSeriesDataType`. Otherwise these calls return an `SDKError`. The `id` is the ID of the object being modified; for the batch version, the two arrays are processed in parallel.

NOTE: Modifications that do not pass the de-duplication system will generate the same kind of Duplicate messages as the Registration functions.

Besides the standard errors, Modify can also return `IncompatibleCreationTypes`.

4.2.3 HELPER FUNCTIONS FOR REGISTER AND MODIFY

These functions are useful for converting the XML from the CreateXXXDataType classes into XML suitable for passing to the Registration and Modify interfaces.

```
public static Registration.CreationType
discoverCreationType(Object req)

public static Registration.CreationType
discoverCreationType(Object xml)
```

Both of these return `null` if a `CreationType` cannot be determined.

```
public static String Registration.addCreateHeader(String xml)
```

Adds <Create type="...."> </Create> tags around the XML if it can find a creation type, null otherwise. See the source code for MatchTool and **Error! Reference source not found.** provided with the SDK for examples.

```
public static String Modify.addModifyHeader(String xml, String
id)
```

Uses the input XML and id to generate XML of the following form:

```
<Modify type="...."><ID>id</ID>xml</Modify>
```

if it can find a creation type, returning null otherwise. See the source code for ApplyModBaseTool provided with the SDK for an example.

```
public static String objToModificationXML(Object obj, String
id)
```

Turn the input object into `<Modify>…</Modify>` XML if a creation type can be discovered for the referenced `obj`; `null` otherwise.

4.2.4 MODIFICATION BASE

Record modification requests require a ModificationBase record. These can be retrieved from the Registry using one of two methods:

```
public Response getModificationBase(EIDRConnection eidrConn,
String assetID, CreationType ct)

public Response getModificationBase(EIDRConnection eidrConn,
String assetID, String type)
```

These return object metadata for `creationType`. The object returned is `CreateBasicDataType`; `CreateSeriesDataType`; etc., and the XML is `<Basic>...</Basic>`; `<Series>...</Series>`; etc.

The underlying object must be able to support all the fields needed by the identified `creationType`. It is incorrect to call `GetModificationBase(ID, "CreateSeason")` on an object that was created with `CreateSeries`, for example. In this case, only `CreateBasic` and `CreateSeries` would be permitted.

Errors: standard errors plus `IncompatibleCreationTypes`.

4.2.5 ADD RELATIONSHIP

Any relationship added to a content record must pass both schema validation and the Registry's validation rules. See "Content Record Creation and Modification" in the ***EIDR Registry Technical Overview*** for tables of permitted combinations.

The `org.eidr.sdk.api.AddRelationship` class has the following methods:

```
public Response addSingleRelationshipFromObj(EIDRConnection
eidrConn, String sourceID, Object infoObj, boolean bImmediate)
```

`infObj` is of a type such as `CompositeInfoType` or `PackagingInfoType`. The `sourceID` is the ID of the content record to which the relationship is to be added.

```
public Response addSingleRelationshipFromXML(EIDRConnection
eidrConn, String sourceID, String infoXML, boolean bImmediate)
```

`infoXML` is the XML for <CompositeInfo>...</CompositeInfo>, <PackagingInfo>...</PackagingInfo>, etc. The `sourceID` is the ID of the content record to which the relationship is to be added.

Both of these calls return a status token, as with all batchable operations. If the relationship cannot be added to the object, a `ValidationError` is returned.

The creation type and relationship in the XML version of the call must match when using:

- `CompositeRelationship, compositeInfoType`
- `PackagingRelationship, packaginInfoType`
- `SupplementalRelationship, supplementalInfoType`
- `AlternateContentRelationship, alternateContentInfoType`

4.2.6 REMOVE RELATIONSHIP

Removing a relationship is subject to the Registry's validation rules.
`org.eidr.sdk.api.RemoveRelationship` provides the following methods:

```
public Response removeSingleRelationship(EIDRConnection eidrConn,
String idSource, TargetRelationshipType type, boolean bImmediate )
```

This request, if successful, removes all relationships of the given `type` from `idSource`:

```
public Response removeSingleRelationship(EIDRConnection eidrConn,
String idSource, TargetRelationshipType type, String targetID,
boolean bImmediate )
```

This request, if successful, removes the single relationship described by the pair {`type`, `targetID`} from `idSource`.

```
public Response removeRelationshipBatch(EIDRConnection eidrConn,
string[] idSource, TargetRelationshipType type)
```

This request, if successful, removes all relationships of a `type` from the content IDs in the array `idSource`.

```
public Response removeRelationshipBatch(EIDRConnection eidrConn,
RemoveRelationshipType removals[])
```

This request, if successful, performs a fully general relationship removal, allowing a mix of removal with and without targetID, and a mix of relationship types. `RemoveRelationshipType`, which identifies the type that will be removed by `removeRelationshipBatch`, has three fields:

- **ID** (The ID from which to remove the relationship.)
- **TargetID** (The optional target of the relationship to be removed. If present, it removes only a relationship of `type` with a target of `targetID`. If not present, the operation removes all relationships of the given `type` from the record identified by `ID`.)
- **Type** (The type of relationship to remove.)

`removeSingleRelationship` and `removeRelationshipBatch` return a status token, as with all other batchable operations. If the relationship cannot be removed from the object, the Registry returns a `NotRemovable` error.

4.2.6.1 TIPS AND TRICKS

The `IsCompositeOf` relationship is only removable if the object's referent type is not `Composite`.

`TargetID` is allowed only for lightweight relationships.

4.2.7 ALIAS

One of the fundamental goals of the EIDR Registry is to have only a single ID for a particular piece of content. (See "Aliases and Deletion Model" in the ***EIDR Registry Technical Overview***.) This is enforced by the de-duplication system, which uses both automated systems and human review for identification of potential duplicates. Sometimes, however, a content record is registered that later turns out to be a duplicate of an already-existing record. It would be inappropriate to just delete the duplicate record, since it may have already been promulgated to third parties. Instead, EIDR supports aliasing an ID to a different content record.

Only an object with no relationships and no dependents can be aliased. Use `FindDescendents()` and `GetLightweightRelationships()` (or resolve to `simpleDataInfoType`) to find the relationships and `Get Dependents()` for dependent items. When an ID is aliased:

- All resolutions related to the old ID will return data from the new content record, unless the resolution has set the `followAlias` flag to `false`. (See the "Resolution" section for details.)
- All other calls related to this ID return an error. Applications should follow the alias chain to get the definitive EIDR ID, and then use it in place of the aliased ID in the future. For example, an aliased ID cannot be used as the ID in a rooted query, nor can it be passed in to a GraphTraversal. Most EIDR calls will return an AliasNotAllowed error, although some may return BadID.
- The `LastModificationDate` field in the Provenance record for an aliased ID is the time at which the ID was aliased. Users may not further modify an aliased record.

The `org.eidr.sdk.api.Alias` class aliases an ID to a different record. This class two methods:

```
public Response aliasSingle(EIDRConnection eidrConn, String id,
String target, boolean bImmediate)

public Response aliasBatch(EIDRConnection eidrConn, string[] ids,
string[] targets)
```

In both of these calls, the both the ID and target must be valid EIDR content IDs. Both of these calls return a status token, as with all batchable operations.

Errors: standard errors plus `HasDependents` and `IncompatibleAliasTarget`.

4.2.7.1 TIPS AND TRICKS

See the "Resolution" section for ways of determining whether an ID has been aliased.

The ForceDedupe flags do not apply to this call.

4.2.8 DELETE

EIDR IDs are permanently resolvable, as are all DOIs (of which EIDR IDs are an implementation). In other words, once an ID has been issued, it persists in the Registry and can always be resolved. To ensure this, EIDR IDs are never physically deleted. Instead, deleted IDs are just aliased to the EIDR "tombstone" record. See the ***EIDR Registry Technical Overview*** for more information.

The `org.eidr.sdk.api.Delete` class has two methods:

```
public Response deleteSingle(EIDRConnection eidrConn, String id,
boolean bImmediate)

public Response deleteBatch(EIDRConnection eidrConn, string[] ids)
```

The response to these requests contains a status token, as with all batchable operations.

If a Delete request is successful, the `LastModificationDate` field in the Provenance for the ID will be set to the time at which the ID was aliased to the tombstone record.

Only an object with no relationships and no dependents can be deleted. Use `FindDescendents` and `GetLightweightRelationships` (or do a Simple resolution) to find any relationships and `GetDependents` to find dependent records. Each defined relationship must be removed before an object may be deleted.

Errors: standard errors plus dependency errors.

4.2.8.1 TIPS AND TRICKS

The ForceDedupe flags do not apply to this call. See the ***EIDR HTTP API Reference*** for more information about error handling.

4.2.9 PROMOTE

An “in development” content record can be promoted to be a “valid” object. “In development” objects are not subject to de-duplication review because they are private to the registrant. This means they are not compared to other records in the registry at the time of their creation or in the future when other, potentially similar, records are created or modified. They are only checked for duplicates when they are promoted to “valid” objects.

The `org.eidr.sdk.api.Promote` class has two methods:

```
public Response promoteSingle(EIDRConnection eidrConn, String id,
boolean bImmediate)

public Response promoteBatch(EIDRConnection eidrConn, string[] ids)
```

The response to these requests contains a status token, as with all other batchable operations. The status may indicate success or the reason that the candidate was not promoted (typically, because it failed to pass de-duplication review).

If Promote is successful:

- The content record’s Status is set to "valid".
- The `LastModificationDate` field in the provenance record for the promoted ID is set to the time at which the object was promoted.
- The Promote and View ACLs are removed from the object, and it becomes viewable by any user

Errors: Standard errors plus `NotInDev`.

4.2.9.1 TIPS AND TRICKS

Because of their special de-duplication handling, it is possible for other records to be registered that duplicate records in “in development” status. To avoid this, in development records should be used only when absolutely necessary and then promoted to valid status as soon as possible.

4.3 TOKEN STATUS LOOKUP

`StatusLookup` requests return information about previous registry operations using their associated batch, operation, or user tokens. The information returned may change in the future if the token is not presently in a terminal state.

4.3.1 TERMINAL STATES

4.3.1.1 OPERATION TOKENS

The OperationStatus/Status element returned by StatusLookup operation will not change once the processing of the associated registry operation has reached a terminal state. Anything other than

Pending is a terminal state. OperationStatus/Status/Code is numeric from 0-6, and corresponds to the OperationStatus/Status/Type states listed below. (These are defined in detail in api-common.xsd).

OperationStatus Code	OperationStatus Type
0	success
1	duplicate
2	pending
3	authorization error
4	validation error
5	other error
6	rejected

Usually, if a non-Success result is returned by a request, the request needs to be corrected before it is resubmitted. If a transient error is returned, the problem is not with the request, but with communications with the Registry or operations within the Registry.

- **Duplicate Error** (The metadata in the request was found to be duplicative of the metadata in other object(s) in the registry. The request should not be resubmitted until the metadata in the submitted record or the identified duplicates have been changed – unless there is certainty that the submitted record is unique despite the reported duplicate(s), in which case it can be submitted for forced manual review.)
- **Authorization Error** (The credentials in the request were not appropriate given the requested operation, the requester's allowed roles, and/or the ACL of the objects involved. The request should not be resubmitted until one of the above security conditions has changed.)[11]
- **Validation Error** (The metadata in the request or the related objects did not conform to the XML schema specification or the Registry's additional data validation rules. The request should not be resubmitted until the affected metadata has changed)
- **Rejected Error** (A request has been rejected during manual de-duplication review without being processed. This is very rare. The request should not be resubmitted without first consulting with EIDR support.)

[11] A very small number of Authorization Errors may actually be transient, but this is extremely rare and cannot be distinguished from a true authentication error.

- **Other Error** (Returned for various transient problems, like bad communication with the de-duplication system, and may be resubmitted. Since it may reflect some other error, and "transient" does not necessarily mean short-lived, caution should be used before submitting a third time.)

4.3.1.2 BATCH TOKENS

The following applies only to batches containing more than one item.

Once a batch has passed top-level authentication, syntax checking, and so on, these are the possible states:

- **1** (batch received) The batch itself has passed validation and is being turned into individual requests subject to further validation. No further information is available at this point.
- **2** (batch queued) The individual requests have passed validation and have all been submitted. StatusLookup requests may now be used with the batch token return the individual tokens and the current state of each individual request.
- **3** (invalid batch) This can result from a bad user token. This can also result from abnormal operation of the Registry, which should be reported to EIDR support.

Batch queued and invalid batch are terminal states for a batch token.

4.3.2 MATCH SCORES

Scores can be returned in response to immediate-mode Match requests, to give an idea of how close any current Registry records are to the contemplated registration. Scores are meaningful *only* for immediate-mode Match requests. If they are present in the response to a non-immediate (asynchronous) request, they should be ignored.

4.3.3 POLLING

Poll (request the current status of) a token at intervals using StatusLookup until it reaches one of the possible end states. For a batch token, extract all the operation tokens once the batch has reached the "batch queued" state[12] and manage them all individually, or you can poll on the Batch Token, dealing with each operation token as it reaches an end state or after all of them have reached an end state. The former is usually somewhat better, since you do not need to continually poll on the Batch Token; the latter is less efficient but may be preferable when you do not want the complexity of managing multiple tokens.

For operation tokens, poll them individually, one at a time, unless they are part of a batch, when they can be polled indirectly via the batch token..

---

[12] It is not sufficient to check for Success in `/Response/Status` field – that just means the request was accepted by the Registry.

4.3.4 STATUSLOOKUP

The `org.eidr.sdk.api.StatusLookup` class has the following methods:

- `Response statusLookupViaRegistrant(EIDRConnection eidrConn, java.lang.String registrant, int pageNumber, int pageSize)`
- `Response statusLookupViaRegistrant(EIDRConnection eidrConn, String registrant, int pageNumber, int pageSize, java.util.Date from, Date to, java.lang.String status)`
- `Response statusLookupViaRegistrant(EIDRConnection eidrConn, String registrant, String pageNumber, String pageSize)`
- `Response statusLookupViaToken(EIDRConnection eidrConn, String token, int pageNumber, int pageSize)`
- `Response statusLookupViaToken(EIDRConnection eidrConn, String token, String pageNumber, String pageSize)`
- `Response statusLookupViaUser(EIDRConnection eidrConn, String user, int pageNumber, int pageSize)`
- `Response statusLookupViaUser(EIDRConnection eidrConn, String user, int pageNumber, int pageSize, Date from, Date to, String status)`
- `Response statusLookupViaUser(EIDRConnection eidrConn, String user, String pageNumber, String pageSize)`

To retrieve the status of a recent request, invoke the `getStatus()` method of the `Response` object. An `org.eidr.sdk.model.StatusTypes` enumeration value is returned, indicating success or an error.

One of the `GetStatus` methods can be used to query the status of the request.

```
GetStatus(token, pageNumber, pageSize, continuationToken)
```

Errors: standard errors plus `BadQuery` and `ResultTooLong`.

The paging parameters are used as follows:

- Results are returned with `pageSize` objects per batch.
- The `pageNumber`-th page of `pageSize` records is returned. Page 1 is the first page. Page 0 is a request for all results.
- `ContinuationToken` should be a value returned by the previous paged GetStatus request using this query string, or null if this is the first call in the set.

Returns `(token, status)` elements if `token` is not a batch token.

If `token` is a batch token, returns:

- **`PageSize`** (Max number of records included in a single response page – equal to input parameter `pageSize`.)
- **`MySize`** (The size of this chunk – equal to `pageSize`, except perhaps for the last page.)

- **`TotalPages`** (The total number of response pages that can be returned given the `pageSize`. Returned – equal to $\left\lceil \frac{TotalResults}{PageSize} \right\rceil$.)
- **`PageNum`** (The number of the current returned page.)
- **`TotalResults`** (The total number of result records across all pages.)
- **`ContinuationToken`** (A simple string marking the current position within the results. Must be passed as the `ContinuationToken` to a new `GetStatus` request to obtain the next results page in sequence.)
- A list of the `(token, status)` elements for each item in the request response that falls in the current returned page of results.

Response statuses for a batch `GetStatus` request:

- **`BatchReceived`** (The batch has been fully read by the Registry, but no status for its individual sub-elements is yet available.)
- **`BatchQueued`** (The batch is being processed by the Registry. Each sub-element in the batch now has an available status that may be queried.)
- **`BatchInvalid`** (There was a Registry error processing the batch; one of the standard errors is also returned as a detail.)

Response statuses for the individual sub-elements in the batch:

- **`Success`** (The request has succeeded. For example, by creating a new object or deleting an old one. The detail field contains the EIDR ID associated with the successful operation: a new EIDR ID for creation, and the passed-in EIDR ID in other cases.)
- **`Duplicate`** (When a duplicate or potential duplicates are found during create, modify, or promote. The detail field contains one or more EIDR IDs of items detected as possible duplicates; the IDs for all duplicates are returned.)
- **`Pending`** (When the Registry is waiting to finish an operation, but has not yet had any errors; in the case of create, modify, and promote it may have as a detail a list of EIDR IDs for one or more potential duplicates.)
- **`Failure`** (When the request fails. For example, after an attempt to create a duplicate item.)

- **`Error`** (Adds one or more of the error values as a detail.)

Errors: any of the standard errors or a `BadToken` error.

```
GetStatus(Registrant, filter, token, page, pageSize,
continuationToken)

GetStatus(User, filter, token, page, pageSize,
continuationToken)
```

The last four arguments are treated as they are for other `GetStatus` requests.

`Filter` can be:

- **`Status`** (*`legal-status-value`*: Only tokens that match the supplied value are returned.)
- **`After`** (*`date-time`*: Only tokens issued after the supplied time are returned.)
- **`Range`** (*`date-time-start date-time-end`*: Only tokens generated between the two dates are returned.)

Legal combinations are:

- No filter
- Status only
- After only
- Range only
- Status and After
- Status and Range

A `Registrant` request returns a paginated list of `(token, status)` records for every token belonging to `Registrant` that matches `Filter`. A token belongs to a Registrant if a user in the `Registrant` Party made the request with which it is associated.

`User` returns a paginated list of `(token, status)` for every token generated by a request from `User` and matching `Filter`.

5 PARTY RECORD APIs

A party is an organization uniquely identified within the EIDR registry and assigned an EIDR Party ID, such as record Registrants, Associated Orgs, and Metadata Authorities.

5.1 RESOLVE

With the Java and .NET SDKs, to make a resolve request, instantiate an `org.eidr.sdk.api.Resolution` object and call one of the `resolve` methods:

- `KernelMetadataResponse resolveDOIKernel(EIDRConnection eidrConn, String assetID, boolean followAlias)`
- `FullObjectInfoResponse resolveFull(EIDRConnection eidrConn, String assetID, boolean followAlias)`
- `ProvenanceInfoResponse resolveProvenance(EIDRConnection eidrConn, String assetID, boolean followAlias)`
- `AllSelfDefinedInfoResponse resolveSelfDefined(EIDRConnection eidrConn, String assetID, boolean followAlias)`
- `SimpleInfoResponse resolveSimple(EIDRConnection eidrConn, String assetID, boolean followAlias)`

The call for making a party resolve request is in org.eidr.sdk.party.PartyResolution and is as follows:

```
PartyResolutionResponseType
partyResolve(EIDRConnection eidrConn, PartyResolutionTypeEnum type,
String partySuffix)
```

The return type will vary with the member function called. For partyResolve, this is the metadata for the party; the format of the result is dependent on the `type` parameter.

Type	Format
`DOIKernel`	DOI kernel metadata for a Party
`Full`	`eidr:partyResolutionType`; see the schema. `The PartyAccountName` field will match the suffix of the DOI

Errors: Standard errors plus `BadParty`

5.2 PARTY QUERY

With the Java and .NET SDKs, to make a party query request, instantiate an `org.eidr.sdk.party.PartyQuery` object and call one of the following methods:

- `PartyQueryResponse expressionPartyQuery(EIDRConnection eidrConn, ExpressionPartyQueryType request, boolean bFull)`
- `PartyQueryResponse findPartiesByName(EIDRConnection eidrConn, FindPartiesByName request, boolean bFull)`

- `PartyQueryResponse findPartiesFromCatalog(EIDRConnection eidrConn, PartyQueryType request, boolean bFull)`

The call for making a party query request is as follows:

```
PartyQueryResponse findPartiesByName(EIDRConnection eidrConn,
PartyQueryType request, boolean bFull)
```

PartyQueryResponse contains the following member functions in its PartyResponseQueryType::

- **`getTotalMatches`** (Total number of available results.)
- **getCurrentSize** (Returns the `currentSize` property.)
- **`getContinuationToken`** (Returned by implementations that internally cache query results. It should be passed in as the token argument on subsequent calls as an accelerator.)
- A list of results will be returned by getParty() and getParty() that fulfills the query request.

Errors: standard errors plus `ResultTooLong`.

Internally, the Registry keeps a list of Parties sorted by `SortName` (or `DisplayName` if SortName is not present). Sections of this sorted list are returned using the `FindPartiesFromCatalog` call.

6 VIDEO SERVICE RECORD APIs

A video service is an audiovisual content provider, such as a broadcast network or channel, cable TV channel, satellite TV channel, video-on-demand (VOD) provider, etc., identified by a unique ID.

6.1 READ

6.1.1 VIDEO SERVICE RESOLUTION

To make a video service resolution request with the SDK, instantiate an `org.eidr.sdk.service.ServiceResolution` object and call the following method:

```
ServiceResolutionResponseType
serviceResolve(EIDRConnection eidrConn,
PartyResolutionTypeEnum type, String serviceID, boolean followAlias)
```

6.1.2 VIDEO SERVICE QUERY

To make a video service query request with the SDK, instantiate an `org.eidr.sdk.service.ServiceQuery` object and call either of the following methods:

```
ServiceQueryResponse findServicesByName(EIDRConnection eidrConn,
ServiceQueryType request, boolean includeAltNames, boolean bFull)
```

or

```
ServiceQueryResponse
findServicesFromCatalog(EIDRConnection eidrConn,
ServiceQueryType request, boolean bFull)
```

6.2 TREE TRAVERSAL

6.2.1 GET SERVICE PARENT

To get a service parent, instantiate an `org.eidr.sdk.service.ServiceGraph` object and call the following method:

```
Response getServiceParent(EIDRConnection eidrConn,
String serviceID, boolean includeInactive)
```

6.2.2 GET SERVICE CHILDREN

To get a service child, instantiate an `org.eidr.sdk.service.ServiceGraph` object and call the following method:

7 SDK COMMAND-LINE TOOLS

7.1 INTRODUCTION AND PURPOSE

With the Java and .NET SDKs, you can create applications that make use of Registry services and perform Registry operations. The SDKs come with numerous command-line tools that are useful to non-programmers for ad hoc actions or shell-scripted automations and also serve as programming examples for SDK usage, since they all come with full source code and documentation. With the command-line tools you can:

- Resolve EIDR IDs
- Examine object hierarchies
- Make queries about Content Records, Parties, and Video Service Providers
- Create and modify objects
- Manage relationships between objects
- Promote objects[13]
- Delete objects
- Alias objects

Detailed instructions and examples for using the command-line tools are available in ***Command-Line Tools Overview***, included in the Doc directory of the SDK installation package.

7.2 AVAILABLE COMMAND-LINE TOOLS

- AddLWTool.exe
- AdminACLTool.exe
- AliasTool.exe
- AltIDtoEIDR.exe
- AltIDTool.exe
- ApplyModBaseTool.exe
- DeleteTool.exe
- FlattenRedefineTool.exe
- FlattenTool.exe
- Gchange.exe
- GetModBaseTool.exe
- GraphTool.exe
- MatchTool.exe
- PartiesFromNames.exe
- PartyTool.exe
- ProcessStrongMatches.exe

[13] Only applies to "in development" content records.

- PromoteTool.exe
- QueryTool.exe
- RegisterTool.exe
- RemoveRelTool.exe
- ResolveMatches.exe
- ResolveTool.exe
- ServiceAliasTool.exe
- ServiceChildrenTool.exe
- ServiceCreateTool.exe
- ServiceDeleteTool.exe
- ServiceModifyTool.exe
- ServiceParentTool.exe
- ServiceResolveTool.exe
- ServiceTool.exe
- StatusTool.exe

For more information, see ***Command-Line Tools Overview***.

8 APPENDIX: SUMMARY OF BATCHABLE CALLS

Call	Size (single/ batch)	Immediate / Non-Immediate (Async)	Notes
Register			
Registration.registerSingleFromObject() Registration.registerSingleFromXML()	single	Parameter	
Registration.registerBatchFromObj() Registration.registerBatchFromXML()	batch	non-immediate	
Modify			
Modify.modifySingleFromXML() Modify.modifySingleFromObj()	single	Parameter	
Modify.modifyBatchFromXML() Modify.modifyBatchFromObj()	batch	non-immediate	
Promote			
Promote.promoteSingle()	single	Parameter	
Promote.promoteBatch()	batch	non-immediate	
Alias			
Alias.aliasSingle()	single	Parameter	
Alias.aliasBatch()	batch	non-immediate	
Delete			
Delete.deleteSingle()	single	Parameter	
Delete.deleteBatch()	batch	non-immediate	

Call	Size (single/ batch)	Immediate / Non-Immediate (Async)	Notes
AddRelationship			
AddRelationship.addSingleRelationshipFromObj AddRelationship.addSingleRelationshipFromXML()	single	Parameter	These calls can add Compositeand all of the lightweight relationships (Packaging, etc.). The SDK has no calls for submitting more than one AddRelationship at a time.
RemoveRelationship			
RemoveRelationship.removeSingleRelationship(EIDRConnection conn, String idSource, TargetRelationshipType type, boolean bImmediate)	single	Parameter	removes all relationships of specified type from single source
RemoveRelationship.removeSingleRelationship(EIDRConnection eidrConn, String idSource, TargetRelationshipType type, String targetID, boolean bImmediate)	single	Parameter	Removes relationship of type with target targetID from single source
RemoveRelationship.removeRelationshipBatch(EIDRConnection eidrConn, string[] idSource, TargetRelationshipType type)	batch	non-immediate	Remove all relationships of type from each ID in idSource[].
RemoveRelationship.removeRelationship(EIDRConnection conn, RemoveRelationshipType removals[])	batch	non-immediate	Fully general relationship removal, allowing mix of removal with and without targetID.

9 APPENDIX: LIST OF ERRORS

Standard request errors are:

- **`Authentication:`** The system could not authenticate the user and/or group in the request.
- **`Authorization`**: The requester is not authorized to perform the action, or access one or more of the DOIs in the request
- **`RegistryReadOnly:`** The Registry is in read-only mode. If the primary system goes down, the mirror systems are still available for reading. Certain system administration tasks (synchronization with a backup, bulk export, etc.) may also have this result.
- **`SystemError`**: The Registry is unable to deal with the request for some internal reason.
- **`InvalidRequest:`** The request URL is invalid, or contains unrecognized or invalid HTTP headers, or contains unrecognized or invalid information.
- **`BadDOI`**: The request contains a DOI that is syntactically correct but has an invalid checksum or is not recognized by the system. Syntactically malformed DOIs generate a `SyntaxError`. The bad DOI is included as a detail.
- **`SyntaxError`**: The XML in the request does not pass schema validation.
- **`ValidationError`**: The XML in the request is syntactically correct but fails other validation rules, or the request would result in an object that violates the validation rules, or the object passed in is inappropriate for the call. The detail field can be:
  - **`IncompatibleCreationTypes`**: A call to `GetModificationBase()` failed because metadata needed for the requested type is not present on the object.
  - **`NotRemovable:`** Returned on failed attempt to remove a relationship. This can occur either when the relationship does not exist, or when its removal would result in an invalid object or hierarchy.
  - **`AliasNotAllowed:`** An aliased DOI was passed to a function that does not accept aliases.
  - **`IncompatibleAliasTarget`**: A DOI is being aliased to an incompatible object.
  - **`NotInDev:`** The call failed because an attempt was made to promote an object that was not in the `InDev` state. Also includes the offending DOI as a detail. See `Promote()`.

Other common errors are:

- **`BadToken:`** The token sent to `GetStatus()` was not recognized.
- **`BadQuery:`** The query string was malformed.
- **`NoParent:`** The object sent to `GetParent()` is itself the root of the tree.
- **`NoChildren:`** The object sent to `GetChildren()` is itself a leaf of the tree.
- **`ResultTooLong:`** A result was too large to fit in a HTTP response. This can be caused by requesting too large a page size in queries (generally > 100).
- **`BadParty`**: The Party ID submitted as part of a request was invalid.
- **`BadUser`**: The User ID submitted as part of a request was invalid.

For more information on EIDR error codes, see “Error Types” in the ***EIDR Registry Technical Overview*** and “Codes and Descriptions” in the ***EIDR HTTP API Reference***.

10 APPENDIX: NOTES ON MODIFYING RECORDS

Modifying objects is done by retrieving an object, modifying it, and re-submitting it to the Registry. Record modification requests require a record subclassed from `ModificationBase`. The `ModificationBase` class has a class factory method, `getModificationBase`, that will return an appropriate Modification base class for the desired operation:

```
getModificationBase(EIDRConnection, ID, CreationType)
```

This method returns object metadata for `creationType`. The schema used for the return is the same as the one used for creating an object of that type.

For example, for an object created using `CreateEpisode`, `GetModificationBase(EIDRConn, ID, CreateBasic)` returns just the fields of the basic object, and `GetModificationBase(EIDRConn, ID, CreateEpisode)` returns all the metadata needed to create an episode.

The underlying object must be able to support all the fields needed by creationType. It is incorrect to call `GetModificationBase(DOI, CreateSeries)` on the object in the previous example.

After the application modifies the metadata returned from `GetModificationBase`(), it submits the modifications by calling one of the methods in `org.eidr.sdk.api.Modify` with the modified metadata (as object or XML) and the same creationType that was passed to `GetModificationBase`(). Modification requests are sent to the de-duplications system and may contain one `ModificationBase` record (immediate or non-immediate) or multiple `ModificationBase` records (non-immediate only).

10.1 HOW TO USE GETMODIFICATIONBASE

If you know that you want to modify a particular field and that the object you have supports it, just use the appropriate creationType. For example, if you want to change a series end date, use CREATE_SERIES as the modification base.

10.1.1 WHEN TO USE CREATE_BASIC

To add an alternate ID, which is in /FullObjectMetadata/BaseObjectdata, you should be able to call getModificationBase(EIDRConn, ID, CREATE_BASIC), extract the object from the response, add the alternate ID, and then call modifySIngleFromObj(EIDRConn, object, CREATE_BASIC).

Alas, there are some caveats in this:

- If the item you want to modify is a root object, `GetModificationBase(…CREATE_BASIC)` returns an object that has CreationFullInfo, which in turn can be sent to `Modify(…, CREATE_BASIC)` – the Registry requires FullInfo to be submitted for creationType CREATE_BASIC.
- If the item is not a root object, inherited fields are not returned and the response will contain `CreationSelfDefinedInfo`. This will only work when passed back to `Modify(…, CREATE_BASIC)` if none of the missing (inherited) fields is a required field. Otherwise, the Registry will return an error because of the “missing'”field.

- The information you get back from any other creation type can always be submitted back to `modify()` using the same creation type.

This all means:

- It is only safe to use `GetModificationBase(…CREATE_BASIC)` if you know the record is a root object or that none of the record's required fields are inherited.
- If you want to modify a base field in a record to which that does not apply, you have to get an appropriate non-basic modification base for the object.
- You are guaranteed to be able to use CREATE_BASIC for Basic and Series objects, but for all others you have to use CREATE_TYPE, where TYPE is the underlying object's inheritance relationship.

All of this makes it tedious to write code that handles general modifications to the base metadata of unknown records – code that wants to modify a base field has to determine whether the record is a root object or not, get the modification base that will work for it, and remember to call `modify()` with that creationType (which may not be CREATE_BASIC (even though the field is on the base object).

The SDK provides some methods to make this easier:

- **`getCreationType`** (Returns the CreationType.)
- **`getModificationObj`** (Returns the modification base object, which can, for example, be passed in to `Modify.modifySingleFromObj`.)
- **`getBaseObjectData`** returns the base object data from `ModificationObj`. If creationType is CREATE_BASIC, the type is CreationFullInfo; otherwise it is CreationSelfDefinedInfo.

As an example, you can get the list of alternate IDs with this (glossing over the error handling), which is the same as the other SDK calls that return Response (for example, `ModificationBase`).

When you call `modify()`, use `resp.getBaseObjectData()` for the object to pass in or convert to XML, and `resp.getCreationType()` for the creation type.

In addition, `org.eidr.sdk.utils.BasInfoConverter` is a utility class that converts back and forth between CreationSelfDefinedInfo and creationFullInfo.

See the source code for AltIDTool and TestBaseMetadataConverter included with the SDK for examples of how to use these classes.

11 APPENDIX: TOKEN USE EXAMPLES

These show the Response XML returned in various circumstances.

11.1 REGISTER ONE ITEM, IMMEDIATE MODE – SUCCESS

This is the result for a Registration. Everything you need is in the initial Response. Note that the RequestStatus token (the "batch token") and the OperationStatus token are the same.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329304802304006432</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329304802304006432</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/6B7E-4CE9-0B43-CAB7-D8C0-2</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

If you look up the token later with StatusLookup, you get the same thing – the operation status is Success, which is a final state.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329304802304006432</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329304802304006432</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/6B7E-4CE9-0B43-CAB7-D8C0-2</ID>
    </OperationStatus>
```

```
    </RequestStatusResults>
  </Response>
```

11.2 REGISTER ONE ITEM, IMMEDIATE MODE – FAILURE

This is also for a registration, but one that fails because the system found a duplicate. (Not surprising, since it is the result of re-running the request from the previous example.) The single RequestStatusResults/OperationStatus/ID element indicates that the ID is an exact match for the attempted registration and can be used as the ID for the submitted object. The threshold information in the <Duplicate> tag shows how closely the request matched the submitted item.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293051732170064 34</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293051732170064 34</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/6B7E-4CE9-0B43-CAB7-D8C0-2</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/6B7E-4CE9-0B43-CAB7-D8C0-2</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

If you were to look the token up again, you would get the same result, since the operation status of duplicate is a final state.

11.3 REGISTER ONE ITEM, NON-IMMEDIATE (ASYNCHRONOUS) – SUCCESS

Note that the RequestStatus token (the “batch token”) and the OperationStatus token are the same.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293056449490064 38</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
```

```
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

The OperationStatus is “batch received”, which is not a terminal state. Depending on timing and load on the Registry, this initial state will sometimes come back as already “pending”. When we query again, the OperationStatus has changed to “success”.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329305644949006438</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329305644949006438</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/B3EC-118C-C235-167F-40CC-L</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

If the registration goes quickly you may see the result with Success as the first response, and never see ”batch received” or “pending“ status.

11.4 REGISTER ONE ITEM, NON-IMMEDIATE (ASYNCHRONOUS) - FAILURE

Here is the initial response. Note that there is a BatchStatus of “batch received”, so this token is acting like a batch token rather than an Operation token.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329305993064006440</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
```

```
      </BatchStatus>
    </RequestStatusResults>
</Response>
```

A later StatusLookup gives us something that looks like it came from an Operation token, because the single-item batch has been queued and there is enough information to fill out the OperationStatus. You may not see this state if there is a sufficient length of time between the initial request and the status lookup, or you may see this as the initial response.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293059930640064400</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293059930640064400</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Since it was still not a final state, we look it up again later. Now the request has failed with an OperationStatus of Duplicate. The `/RequestStatusResults/OperationStatus/ID` field has the same meaning as before; it is a high enough match that it can be used as the ID for the attempted registration

NOTE: For non-immediate (asynchronous) requests, the attributes returned in RequestStatusResults/OperationStatus/Duplicate *should not be used*. They will not always be present, and when present may be incorrect.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293059930640064400</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293059930640064400</Token>
```

```
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/B3EC-118C-C235-167F-40CC-L</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/B3EC-118C-C235-167F-40CC-L</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.5 REGISTER TWO ITEMS – SUCCESS

Since there is more than one item, this has to be non-immediate (asynchronous).

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329307939921006455</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

Since “batch received” is not an end state, we have to poll (look up the token) again.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
 <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329307939921006455</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1329307940251006456</Token>
```

```
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13293079402520006457</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

The batch is now in a final state ("batch queued") and there are some Operation tokens to look at. Both of them are pending, and so have to be looked up again. Here they are:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293079402510006456</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293079402510006456</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>

<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293079402520006457</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293079402520006457</Token>
```

```
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

After waiting for the token to be resolved by EIDR Operations staff, polling the Operation tokens again gives a final result:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329307940251006456</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329307940251006456</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/B672-7FB2-7609-DBE7-7251-4</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>

<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329307940252006457</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329307940252006457</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/E671-02A8-0E11-295E-7806-0</ID>
    </OperationStatus>
```

```
  </RequestStatusResults>
</Response>
```

Both have succeeded. Now take a look at the batch token again.

NOTE: We can only discern the state of the individual operations from the batch token, not the details of the operations.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293079399210064 55</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13293079402510064 56</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13293079402520064 57</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.6 REGISTER TWO ITEMS – FAILURE

This is pretty straightforward. First reply is the expected “batch received”.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293097104820064 71</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
```

```
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

Then we poll the token again, eventually getting “batch queued”.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293097104820006471</Token>
    <PageNumber>1</PageNumber>
    <PageSize>5000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13293097108860006472</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13293097108860006473</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Both Operation tokens are pending – we must be more patient next time. It may be useful to introduce a delay between token polls, particularly if the transaction has been referred to manual review.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293097108860006472</Token>
    <PageNumber>1</PageNumber>
    <PageSize>5000</PageSize>
  </RequestStatus>
```

```
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329309710886006472</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>

<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329309710886006473</Token>
    <PageNumber>1</PageNumber>
    <PageSize>5000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329309710886006473</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

But eventually they wend their way to a final state. All the comments about the result of a single non-immediate (asynchronous) request apply here too.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329309710886006472</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329309710886006472</Token>
      <Status>
        <Code>1</Code>
```

```
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/B672-7FB2-7609-DBE7-7251-4</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/B672-7FB2-7609-DBE7-7251-4</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>

<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293097108860064 73</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293097108860064 73</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/E671-02A8-0E11-295E-7806-0</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/E671-02A8-0E11-295E-7806-0</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

As with the previous non-immediate (asynchronous) batch of 2 operations, StatusLookup on the batch token just gives the OperationStatus values and no details. If you saw this result before you had looked up the OperationTokens, you would still have to look them up to get details of the Duplicate IDs.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293097104820064 71</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
```

```
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1329309710886006472</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1329309710886006473</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.7 REGISTER TWO ITEMS – ONE SUCCESS, ONE FAILURE

This example builds upon the previous ones.

Initial response:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329314360824006495</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

StatusLookup on the batch token:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329314360824006495</Token>
    <PageNumber>1</PageNumber>
    <PageSize>5000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
```

```
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13293143614680006496</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13293143614690006497</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Status lookup on the OperationTokens.

This one has failed already:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293143614680006496</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293143614680006496</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/B672-7FB2-7609-DBE7-7251-4</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/B672-7FB2-7609-DBE7-7251-4</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

But this one is still pending.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
```

```
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293143614690064 97</Token>
    <PageNumber>1</PageNumber>
    <PageSize>5000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293143614690064 97</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Trying just the pending one again, we see it has succeeded.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293143614690064 97</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13293143614690064 97</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/F345-0ACE-8557-9EBE-5B68-P</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

And see the final states in the two OperationTokens in the StatusLookup of the BatchToken.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13293143608240064 95</Token>
```

```
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13293143614680064 96</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13293143614690064 97</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.8 DELETE SINGLE ITEM – SUCCESS

Responses to Delete are like those for Register, but with fewer possible error conditions. This response came from a single immediate-mode request, but could just as well be the lookup of the token for a single-item batch (as was the case for Registration).

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13259362732170058 67</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>13259362732170058 67</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/B38A-47F0-786F-52E4-032E-Q</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.9 DELETE SINGLE ITEM – FAILURE

The OperationStatus `<Details>` field here is a little misleading – this was an attempt to Delete an already deleted item – but the structure of the response should have no surprises for the attentive reader.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325936337610005869</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1325936337610005869</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>ID must be an identifier of a valid asset</Details>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.10 DELETE MULTIPLE – SUCCESS

This is similar to a successful multiple Registration. First the batch is received:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325935799747005861</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

Then it is queued and gives us a set of OperationTokens:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
```

```
  <RequestStatus>
    <Token>1325935799747005861</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>4</CurrentSize>
    <TotalMatches>4</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1325935800033005862</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005863</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005864</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005865</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Since even the most dedicated reader may be flagging by now, here is the lookup of just one of the OperationTokens:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325935800033005862</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
```

```
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1325935800033005862</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Alas, still pending. Now look at the BatchToken again:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325935799747005861</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>4</CurrentSize>
    <TotalMatches>4</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1325935800033005862</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005863</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005864</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1325935800034005865</Token>
      <Status>
        <Code>0</Code>
```

```
        <Type>success</Type>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

They have all succeeded, and here is the StatusLookup of one of them. Note that unlike the lookup of the BatchToken, it gives you the ID that was deleted as part of the OperationStatus:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325935800033005862</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1325935800033005862</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/EA58-A374-0E79-7C7C-9110-G</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.11 DELETE MULTIPLE – FAILURE

This is what happens when a batch of Delete requests fails:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325935545881005856</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

They have all gotten to an end state rather quickly – validation rule failures are faster to the finish line than full processing of a valid request.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13259355458810058 56</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>3</CurrentSize>
    <TotalMatches>3</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13259355464450058 57</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>ID must be an identifier of a valid asset</Details>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13259355464460058 58</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>ID must be an identifier of a valid asset</Details>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>13259355464460058 59</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>ID must be an identifier of a valid asset</Details>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

Looking up one of the tokens, it looks just like the Response for the failure of a single Delete request.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13259355464460058 58</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
```

```
      <CurrentSize>1</CurrentSize>
      <TotalMatches>1</TotalMatches>
      <OperationStatus>
        <Token>1325935546446005858</Token>
        <Status>
          <Code>4</Code>
          <Type>validation error</Type>
          <Details>ID must be an identifier of a valid asset</Details>
        </Status>
      </OperationStatus>
    </RequestStatusResults>
  </Response>
```

11.12 DELETE MULTIPLE – MIXED RESULTS

For completeness, here is a mixed-result batch Delete. First the "batch received" state:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13259365136260005875</Token>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <BatchStatus>
      <Code>1</Code>
      <Type>batch received</Type>
    </BatchStatus>
  </RequestStatusResults>
</Response>
```

And now we get the OperationTokens once the batch is queued:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>13259365136260005875</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>3</CurrentSize>
    <TotalMatches>3</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>13259365141600005876</Token>
      <Status>
```

```
            <Code>2</Code>
            <Type>pending</Type>
          </Status>
        </OperationStatus>
        <OperationStatus>
          <Token>1325936514160005877</Token>
          <Status>
            <Code>2</Code>
            <Type>pending</Type>
          </Status>
        </OperationStatus>
        <OperationStatus>
          <Token>1325936514160005878</Token>
          <Status>
            <Code>4</Code>
            <Type>validation error</Type>
            <Details>ID must be an identifier of a valid asset</Details>
          </Status>
        </OperationStatus>
      </RequestStatusResults>
    </Response>
```

Here are all the individual tokens pending:

```
    <Response xmlns="http://www.eidr.org/schema/" version="2.0">
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <RequestStatus>
        <Token>1325936514160005876</Token>
        <PageNumber>1</PageNumber>
        <PageSize>1000</PageSize>
      </RequestStatus>
      <RequestStatusResults>
        <CurrentSize>1</CurrentSize>
        <TotalMatches>1</TotalMatches>
        <OperationStatus>
          <Token>1325936514160005876</Token>
          <Status>
            <Code>2</Code>
            <Type>pending</Type>
          </Status>
        </OperationStatus>
      </RequestStatusResults>
    </Response>
    <Response xmlns="http://www.eidr.org/schema/" version="2.0">
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <RequestStatus>
        <Token>1325936514160005877</Token>
        <PageNumber>1</PageNumber>
        <PageSize>1000</PageSize>
      </RequestStatus>
      <RequestStatusResults>
```

```
      <CurrentSize>1</CurrentSize>
      <TotalMatches>1</TotalMatches>
      <OperationStatus>
        <Token>1325936514160005877</Token>
        <Status>
          <Code>2</Code>
          <Type>pending</Type>
        </Status>
      </OperationStatus>
    </RequestStatusResults>
  </Response>
  <Response xmlns="http://www.eidr.org/schema/" version="2.0">
    <Status>
      <Code>0</Code>
      <Type>success</Type>
    </Status>
    <RequestStatus>
      <Token>1325936514160005878</Token>
      <PageNumber>1</PageNumber>
      <PageSize>1000</PageSize>
    </RequestStatus>
    <RequestStatusResults>
      <CurrentSize>1</CurrentSize>
      <TotalMatches>1</TotalMatches>
      <OperationStatus>
        <Token>1325936514160005878</Token>
        <Status>
          <Code>4</Code>
          <Type>validation error</Type>
          <Details>ID must be an identifier of a valid asset</Details>
        </Status>
      </OperationStatus>
    </RequestStatusResults>
  </Response>
```

Here is the batch when they are all in a final state:

```
  <Response xmlns="http://www.eidr.org/schema/" version="2.0">
    <Status>
      <Code>0</Code>
      <Type>success</Type>
    </Status>
    <RequestStatus>
      <Token>1325936513626005875</Token>
      <PageNumber>1</PageNumber>
      <PageSize>100</PageSize>
    </RequestStatus>
    <RequestStatusResults>
      <CurrentSize>3</CurrentSize>
      <TotalMatches>3</TotalMatches>
      <BatchStatus>
        <Code>2</Code>
        <Type>batch queued</Type>
      </BatchStatus>
      <OperationStatus>
        <Token>1325936514160005876</Token>
        <Status>
```

```
            <Code>0</Code>
            <Type>success</Type>
          </Status>
        </OperationStatus>
        <OperationStatus>
          <Token>1325936514160005877</Token>
          <Status>
            <Code>0</Code>
            <Type>success</Type>
          </Status>
        </OperationStatus>
        <OperationStatus>
          <Token>1325936514160005878</Token>
          <Status>
            <Code>4</Code>
            <Type>validation error</Type>
            <Details>ID must be an identifier of a valid asset</Details>
          </Status>
        </OperationStatus>
      </RequestStatusResults>
    </Response>
```

Here is more detail in the two successful OperationTokens:

```
    <Response xmlns="http://www.eidr.org/schema/" version="2.0">
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <RequestStatus>
        <Token>1325936514160005876</Token>
        <PageNumber>1</PageNumber>
        <PageSize>100</PageSize>
      </RequestStatus>
      <RequestStatusResults>
        <CurrentSize>1</CurrentSize>
        <TotalMatches>1</TotalMatches>
        <OperationStatus>
          <Token>1325936514160005876</Token>
          <Status>
            <Code>0</Code>
            <Type>success</Type>
          </Status>
          <ID>10.5240/FA9C-CEEA-9DF0-0A43-DDE5-Q</ID>
        </OperationStatus>
      </RequestStatusResults>
    </Response>
    <Response xmlns="http://www.eidr.org/schema/" version="2.0">
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <RequestStatus>
        <Token>1325936514160005877</Token>
        <PageNumber>1</PageNumber>
        <PageSize>100</PageSize>
      </RequestStatus>
```

```
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1325936514160005877</Token>
      <Status>
        <Code>0</Code>
        <Type>success</Type>
      </Status>
      <ID>10.5240/B943-CC50-E7FC-372B-4B96-5</ID>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

But the OperationToken that had a validation error returns no extra information (which makes it differ slightly from a failed registration, which does provide more information.)

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1325936514160005878</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1325936514160005878</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>ID must be an identifier of a valid asset</Details>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.13 PROMOTE SINGLE – FAILURE

A successful Promote is just like the successful things that have been seen above. Below is an example of the result of an immediate promote of an In Development record that is an exact duplicate of something that already exists as a Valid item. The Response is the same as you would expect from an attempted Registration of the item.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1329315951605006516</Token>
  </RequestStatus>
```

```
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1329315951605006516</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/D6CF-E235-D294-B584-4957-G</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/D6CF-E235-D294-B584-4957-G</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

11.14 PROMOTE MULTIPLE – TWO DIFFERENT KINDS OF FAILURE

Here is a batch with two items to Promote. One is an “in development” record that should not be promoted because it is identical to an already existing valid item and one is an item that is already “valid”. The first one is pending at the first Response and the second is marked as a “validation error”.

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1326042087121005921</Token>
    <PageNumber>1</PageNumber>
    <PageSize>1000</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1326042087598005922</Token>
      <Status>
        <Code>2</Code>
        <Type>pending</Type>
      </Status>
    </OperationStatus>
    <OperationStatus>
      <Token>1326042087598005923</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>Status must be "in development"</Details>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
```

```
</Response>
```

Eventually the pending one ends up in the “duplicate” final state:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1326042087598005922</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>1</CurrentSize>
    <TotalMatches>1</TotalMatches>
    <OperationStatus>
      <Token>1326042087598005922</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
      <ID>10.5240/D6CF-E235-D294-B584-4957-G</ID>
      <Duplicate score="100" lowThreshold="55" highThreshold="85">
        <ID>10.5240/D6CF-E235-D294-B584-4957-G</ID>
      </Duplicate>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```

After which StatusLookup on the BatchToken gives:

```
<Response xmlns="http://www.eidr.org/schema/" version="2.0">
  <Status>
    <Code>0</Code>
    <Type>success</Type>
  </Status>
  <RequestStatus>
    <Token>1326042087121005921</Token>
    <PageNumber>1</PageNumber>
    <PageSize>100</PageSize>
  </RequestStatus>
  <RequestStatusResults>
    <CurrentSize>2</CurrentSize>
    <TotalMatches>2</TotalMatches>
    <BatchStatus>
      <Code>2</Code>
      <Type>batch queued</Type>
    </BatchStatus>
    <OperationStatus>
      <Token>1326042087598005922</Token>
      <Status>
        <Code>1</Code>
        <Type>duplicate</Type>
      </Status>
    </OperationStatus>
```

```
    <OperationStatus>
      <Token>13260420875980059 23</Token>
      <Status>
        <Code>4</Code>
        <Type>validation error</Type>
        <Details>Status must be "in development"</Details>
      </Status>
    </OperationStatus>
  </RequestStatusResults>
</Response>
```