EIDR SYSTEM
VERSION 2.0

Registry User's Guide

2016-04-28

Registry User's Guide.

Feedback on this document can be sent to support@eidr.org

TABLE OF CONTENTS

1 Prerequisites

The purpose of this document is to prepare you to use the EIDR system in your production and development workflows.

This document is intended for EIDR users and developers who are preparing to use the EIDR system.

EIDR recommends that you also read the following documents, which contain more detailed information about the topics covered in this guide:

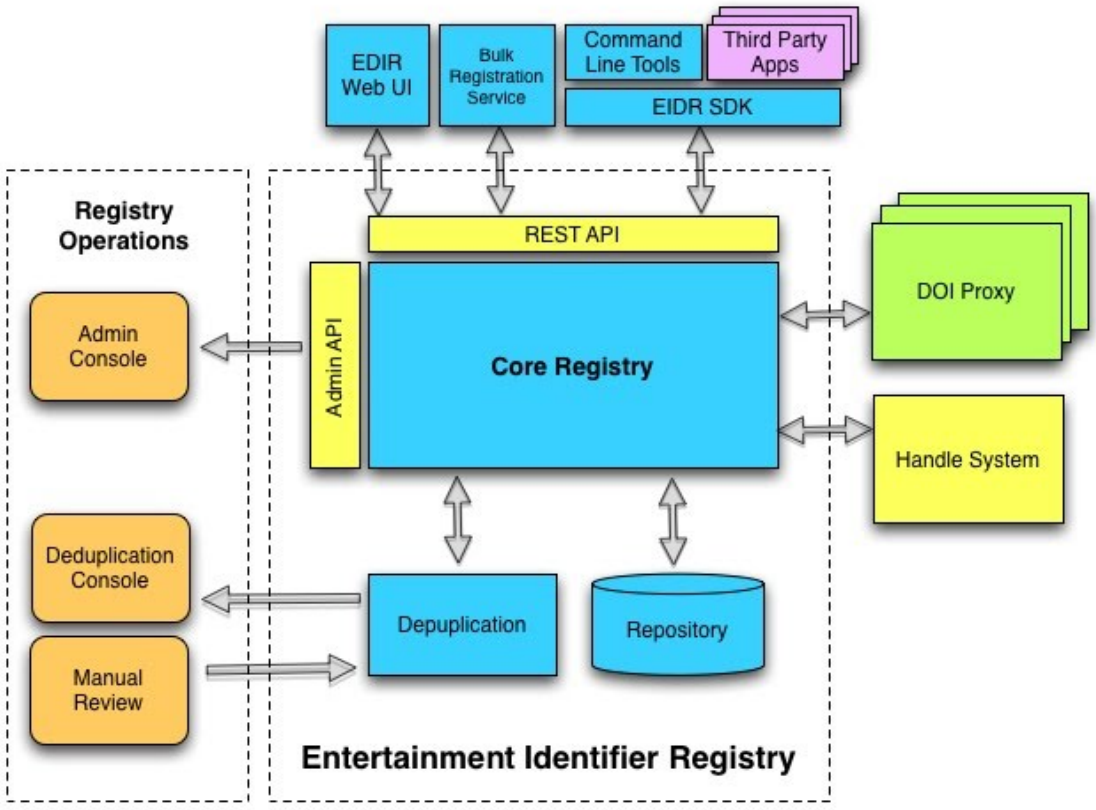
- *Registry Programmer's Guide*
- *Best Practices Guide*
- *Data Fields Reference*
- *REST API Reference*
- *Schema Documents*
- *EIDR Glossary*
- *EIDR: ID Format*

See http://eidr.org/technology/ for additional documentation.

2 Overview

This document describes the usage of the EIDR Registry for EIDR users and developers, including how to read, create, and modify records, how to perform operations on Parties, and how to perform operations on Video Service identifiers. In addition, reference information for the REST API validation rules and the Digital Object Identifier (DOI) Proxy is included.

This diagram illustrates the EIDR system architecture:



Each object in the Registry is assigned a unique, universal, persistent identifier, or EIDR ID. The identifier is a Digital Object Identifier (DOI), a type of Handle conforming to ISO 26324 (for example, 10.5240/0000-0000-0000-0000-0000-X). The EIDR ID is composed of a prefix, which indicates the resolution system (in this case 10.5240 for the EIDR Registry), and a suffix (in this case 0000-0000-0000-0000-0000-X) identifying an object within that system, separated by a slash (/). For more information, see *EIDR: ID Format*.

The EIDR Registry principally identifies commercial audiovisual content. It also identifies important corporate entities in the ecosystem, such as production companies and distribution channels.

The EIDR system consists of the following modules:

Core Registry: This module is a customization and configuration of the Corporation for National Research Initiatives (CNRI) Digital Object Repository. It performs various functions including registration, generation of unique identifiers, indexing, object storage management, and access control. Each object is assigned a unique ID upon registration.

Repository: This stores and provides access to registered objects; for EIDR, these objects are collections of metadata, not the media assets themselves. The metadata includes standard object information, relationships, and access control settings.

Deduplication: This module is called by the Core Registry to check for uniqueness of a newly created or modified object. For more information, see Deduplication.

REST API: A REST interface that provides access to the full set of non-administrative registry features. Using these calls, services can make individual or batched calls, and request immediate rejection or manual review of potential duplicate submissions. A Java SDK, .NET SDK, and sample programs built upon the REST API are available for application and service developers.

Command Line Tools: The command line tools are fairly simple applications, built on the SDK, each of which provide a single function.

Web User Interface: EIDR provides a Web-based user interface primarily for search and lookup. The UI also supports the more common workflows for registration and modification. The Web UI is built with the REST API.

Bulk Registration Service: This application accepts up to 100,000 registration requests at a time and manages submitting them to the registry. It accepts both flat datasets and ones that have internal hierarchies (such as seasons, series, and episodes). This is one of several mechanisms for registering large catalogs after matching has been completed. For more information, see Catalog Matching and Registration.

Handle System: The DOI ecosystem is an application of the Handle system, in the same way that HTTP is built on top of TCP/IP. The Handle system provides distributed lookup and resolution services. The worldwide DOI system is implemented using the Handle system from Corporation for National Research Initiatives (CNRI).

Admin API: This API provides calls to manage accounts, Users, and access control lists (ACLs).

Admin Console: The console enables the registry operator to perform account and user management, access control, and other support functions. It is built using Admin APIs.

Usage Options for the EIDR Registry

You can access the EIDR Registry through its user interfaces, SDKs, and applications such as the bulk registration service and command line tools. The following sections briefly describe these various forms of support.

There are two instances of the EIDR Registry: Production and Sandbox. The Production system is the official site, where all officially registered EIDR IDs exist.

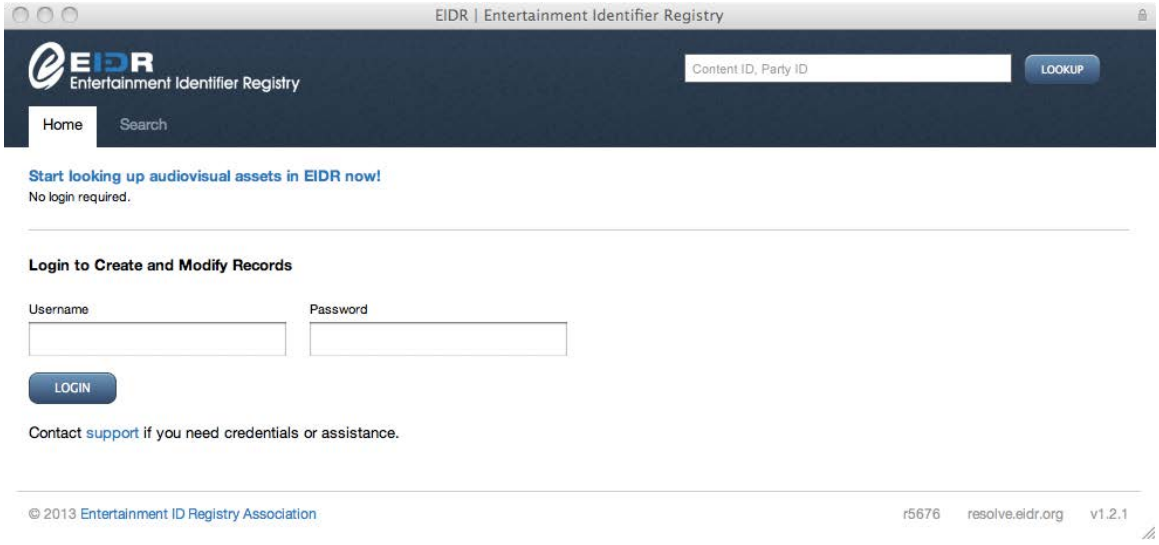
The Sandbox is a development and test site where new users can start their practice registration efforts. Its data will generally not match the production data exactly, and the EIDR IDs do not necessarily match those in the Production system.

Do not assume that the records from the Sandbox are correct, as some of its data are transient. In addition, the Sandbox system is periodically wiped clean and replaced with a copy of the production system as a new starting point, deleting all practice efforts and corresponding EIDR IDs. Notification of such Sandbox refreshes will be sent out to the users, and posted on the EIDR Support Site.

Currently, the EIDR Sandbox Registry UI is available for public access at https://sandboxui.eidr.org. You can search and view currently registered titles, see EIDR numbers of registered titles, and view other metadata needed for uniqueness.

User Interfaces

The user interface is identical between the two registries, so it is important that you use the correct one for your work. Examine the URL or the lower right-hand section of the screen to see the registry and version you are currently viewing.



The production EIDR Registry is available for public access at https://ui.eidr.org/. You can search and view currently registered titles, see EIDR numbers of registered titles, and view other metadata needed for uniqueness.

The publicly accessible Web site has a log in, search options, and link to join EIDR.

Through the user interface you can:

- Perform searches
- Look up the status of asynchronous requests using tokens returned by the requests
- Look up the status of transactions initiated by specific users
- Look up the status of transactions initiated by registrants
- Create records
- Modify records
- Access the EIDR Support network

Command Line Tools

With the command line tools you can:

- Resolve EIDR IDs
- Examine the object hierarchy
- Make queries about Content records, Parties, and Video Service Providers
- Create and modify objects (singly and in bulk)
- Manage relationships
- Promote objects
- Delete objects
- Alias objects

SDK

With the SDK you can create Java and .NET applications that make use of Registry services and perform Registry operations. If necessary, you can use the REST API directly, but this is not recommended.

3 Catalog Matching and Registration

You can match and register back catalogs with EIDR in several ways, but all of them have some common steps. EIDR's process for registering bulk content consists of:

- Preparing data in a standard format, either through XML or a spreadsheet
- An iterative process of matching the proposed records against the current EIDR database
- For records that were not found by the matching process (*gap* records)
    - Serialization: some records (e.g. episodes) may require that other records be registered first (such as registering a series before an episode.)
    - Registration of new content records
- Making any corrections to existing records based on the matching results.
- Providing the matched and newly registered EIDR IDs to the EIDR user for inclusion in the EIDR user's metadata systems.

Most bulk registrations will have to go through the matching process. The matching process reduces the chance of de-duplication errors caused by the variability in quality of source material across providers. In some exceptional cases where it is believed the proposed catalog has minimal overlap with existing EIDR records it is possible to submit a request to the Bulk Registration Service directly.

API integrations should only be used to register new records or records that have already been identified by an external matching effort as *gap* records.

To prepare large registrations:

1. Determine hierarchy compatibility. This includes determining the types of records you will be registering and how these fit with the types of records in your system. For example, if you are registering titles, determine which records represent the original version. If you are registering digital assets, determine those versions (EIDR edits) from which your asset originated.
2. Map your data fields to EIDR's and validate metadata compatibility.
3. Evaluate metadata for covering required EIDR fields and recommended EIDR practices (for example, Directors/Actors and Alternate IDs). Develop a plan to fill in any missing data.
4. Evaluate the quality and consistency of use of the resulting metadata (for example, release year, first billed vs. any two actors).
5. Match production/distribution companies to those in EIDR's Party database.
6. If catalog overlap is expected with existing registrations, use an external matching process before registering the gap records and correcting any identified discrepancies in matched records. If registrations are known to be greenfield (no overlap with existing EIDR registrations), prepare a bulk registration XML file (or use an API integration).

The following describes the steps needed for the matching and registration of titles with the EIDR service.

Data Model Alignment Process

The first step in the process is to make sure that any records being registered align with the EIDR data model.

1. The EIDR user supplies the data model for the records to be registered in the EIDR system.
2. Utilizing the defined and required EIDR data fields need to register a record, the EIDR user iteratively reviews their data model to identify and map to EIDR fields.
3. If needed, the EIDR user can request a review of the data model mapping of EIDR, to help validate that the Provider's data model aligned with EIDR.
4. EIDR user supplies sample set (for example, 10 titles) of data for the fields identified in an Excel template provided by EIDR.
5. EIDR to review this initial template, sample to confirm mapping and gaps.
6. If there are missing data fields or other anomalous issues, EIDR user to iterate with EIDR team on how to provide missing fields.

Output: Mapping of EIDR user's data model and practices to EIDR's.

Record Matching Process

The next step in the process is to match the EIDR user's data against the EIDR data.

1. EIDR will provide a data file (flat file) of the EIDR registry to the EIDR user for matching.
2. EIDR user to review and match titles, and identifies list of titles that are not matched on the EIDR database file.
3. EIDR user inputs missing or gap records it wished to register on an EIDR provided template.
4. Using the provided data, the EIDR team performs a spot check gap records to confirm there are no matches in EIDR.
5. EIDR user then follows the Bulk Registration steps to register records, See steps below.

Output: Gap titles identified and verified for Bulk Registration process.

Party Matching (Production/Distribution Companies)

The use of Parties for production companies is optional but recommended. Once the EIDR user's data model is aligned with EIDR, and all needed data fields for a registration are available, the next step is to confirm all Production and Distribution Parties are present in EIDR. This step can be done in parallel with the data model alignment.

1. The EIDR user builds list of Parties (Production and Distribution companies) they will need as part of the registration process.
2. EIDR gives the EIDR user a set of possible matches for each party in the list

3. For each name in its list, the EIDR user either chooses one of the Parties from the set of choices, or marks the Party as "new".
4. The registrant submits a request to register the "new" Parties.
5. Optionally, a EIDR user can request that a new name be added to an existing Party record

Output: Mapping of the EIDR user's producer/distributor database to EIDR's.

Manual Bulk Registration Process (Not through integrated APIs)

After Data Model Alignment, Record Matching, and Party Matching have been completed, the registration process can start.

1. The EIDR user provides a full set of data fields required for the gap titles identified in the output of the Matching Process in the template provided.
2. EIDR spot checks records:
    a. Identifies records with missing data fields:
        i. For records with missing data, those records are reviewed and missing data manually added.
        ii. In some cases, classes of records may be deferred for later registration.
    b. Spot check data quality for those records with full data fields:
        i. Of the records spot checked, errors are manually addressed. If there is a high percentage of records in error during the spot check, a more thorough review will be needed, and the EIDR user must perform a manual addition of missing data.
3. Once a final data file is provided to EIDR, it is converted utilizing an EIDR provided tool into XML.
4. The XML file is ingested into the EIDR sandbox.
5. After ingest, the EIDR user and EIDR spot check the data and fields:
    a. Errors corrected in input files if possible; otherwise, record made for manual correction.
6. Once quality is confirmed, modified ingest files are sent to the Registry.
    a. EIDR performs any manual fixes noted in 5a.
    b. EIDR spot checks the Registry.

Output: Gap data added to EIDR database.

Note: The EIDR user is then responsible for adding the newly created EIDR content IDs to its data systems.

Bulk Modification Process

1. After Record Matching, if the EIDR user has additional fields such as alternate IDs or more accurate metadata, the EIDR user supplies a CSV file with the EIDR IDs and additional metadata fields.
2. EIDR spot checks metadata for accuracy and alternate IDs for proper granularity.
3. EIDR updates the records.

4 Record Types

In EIDR there are various metadata fields and relationships. Usually the nature of a record is most strongly determined by the set of relationships it has (such as parental relationships), and those are determined to a great extent by its referent type (for example, Series, TV, Web, or Movie). If they do not have relationships, they are root level records.

A *Party* represents an entity such as a registrant or a producing agent. A *User* is an individual (or an abstract thing that can be treated as an individual.)

All Users are associated with a Party. All Registry requests except for Resolution require authentication information for a User, and all requests allow it.

Only Parties have permissions in the system; a User has all the permissions associated with its parent Party. For more information, see Parties.

Note: Only the Registry Operator has the ability to create or modify Parties or Users.

The DOI prefixes used by the registry are:

- 10.5237 – Parties
- 10.5238 – Users
- 10.5239 – Video Services
- 10.5240 – Content Records

Content Records

This section describes the underlying concepts used to describe content records in the EIDR data model, some example applications of these concepts to movie and television records, and the outline structure of the data model itself. In addition, refer to the *EIDR Best Practices Guide* at http://eidr.org/technology/.

Categorization of Objects

All content objects in EIDR are categorized by their *types* and *relationships*.

TYPE

At an abstract level, EIDR has three general kinds of type:

- *Object Type*: Equivalent to the meaning of object *type* in programming languages, which encapsulates the data fields needed for a particular object
- *Structural Type*: Describes the level of abstractness of an object
- *Referent Type*: Describes the nature of the underlying object

Object Type: This is an extension of the DOI Kernel metadata. When the unqualified word *type* is used in EIDR documents, it refers to these object types. See the appendix in the API Overview for details of mapping EIDR fields to DOI fields.

There are ten different content record types in the Registry, which are divided into two general classes:

- **Basic Type**: This type covers the minimal possible object. It is sufficient for describing a wide variety of content.
- **Derived Types:** These types include all the information in the Basic Type, and add extra information for describing more complex objects. The nine Derived Types in EIDR are Edit, Series, Season, Episode, Composite, Compilation, Clip, Manifestation, and InteractiveMaterial.

The DOI specification provides and requires two other kinds of type – Structural Type and Referent Type. Both of these are represented as Basic metadata fields:

Structural Type: This is based on a set of four particular structural types provided by DOI. They correspond to increasingly more specific manifestations of a work.

Structural Type	Use
Abstraction	Used for objects having no reality, such as a series container or the most basic concept of the original work.
Performance	Used for items that are particular manifestations or versions of something, such as the Director's Cut of a film or the Welsh-language version of a TV show.
Digital	A particular digital manifestation of a work, such as an MPEG-2 encoding of a movie.
Physical	A physical version of an object. EIDR will support this for physical films and tapes in a future release

Referent Type: In DOI terms, the referent is the item to which the DOI refers and is independent of any particular instantiation. The DOI handbook says, "referentType typically describes the abstract nature of the content of a referent irrespective of its structuralType". For example, an object created as a movie is a movie whether it is being shown in a cinema, broadcast as an edited version over terrestrial TV, or streamed over the Internet.

Referent Type	Use
Series	An Abstraction that contains ordered or unordered individual items.
Season	A second level of grouping below a Series.
TV	Content that first appeared via broadcast.
Movie	Long-form content that first appeared in a theater (in the US) or a cinema (in most of the rest of the world).
Short	Loosely defined to cover a work that is 40 minutes or less, such as music videos, theatrical newsreels, or theatrical or DTV cartoon shorts.

Referent Type	Use
Web	Content that first appeared on the Web. This is different from content from elsewhere that has been made available on the Web.
Interactive Material	Content that is not strictly audio-visual. It covers DVD menus, interactive TV overlays, customized players, etc.
Compilation	A grouping of discrete multiple assets such as are found on a home entertainment product.
Supplemental	This type is for secondary content whose primary purpose is to support, augment, or promote other content. Examples include trailers, outtakes, and promotional documentaries ("making of" pieces.)

RELATIONSHIPS

A relationship is a casual term for the way in which two objects are connected. Relationships are described with one or more objects and some metadata. They are classified as:

Inheritance relationships: The object on which the relationship exists can inherit basic metadata fields from the object to which the relationship refers. Only one inheritance relationship may exist on an object. The Inheritance relationships are *isSeasonOf*, *isEpisodeOf*, *isEditOf*, *isManifestationOf*, and *isClipOf*.

Dependence relationships: The objects to which the relationship refers have a strong bearing on the basic nature of the object on which the relationship exists. This means that the objects referred to in the relationship must be taken into account when checking for duplicates when an object is created or modified. The dependence relationships are *isCompositeOf* and *isCompilationOf*

Lightweight relationships: There is no inheritance; the objects to which they refer do not influence the underlying nature of the object on which the relationship exists. These relationships are used primarily when moving around the object tree and connecting object trees to each other. The lightweight relationships are *isPackagingOf*, *isPromotionFor*, *isSupplementTo*, and *isAlternateContentFor*.

There is no structural connection between the Supplemental referent type and the *isSupplementTo* relationship. Although a content record with referent type Supplemental will typically have *isPromotionFor* and *isSupplementTo* to another content record, those relationships can exist on any content record, such as a Clip. Similarly, a Supplemental referent type need not have an *isSupplementTo* relationship to anything.

INHERITANCE

Most objects in the registry are related to each other as nodes in a tree. For example, all of the seasons and episodes of a series form a tree rooted in the series object. The

registry also supports additional non-parental relationships, such as one object being included in a composite with items from outside its own hierarchy.

Items in a tree can inherit certain fields from their parent. See the *Data Fields Reference* for full descriptions of these fields. Only metadata from the Basic Type can be inherited. Furthermore, an object can only be part of one tree, so it has only a single chain of inheritance. Lightweight and dependence relationships allow records to interact with objects external to their own hierarchy.

This worldview uses standard computer science terminology: ancestors and descendants, root objects and leaf nodes. Items with both ancestors and descendants are called *internal nodes*.

DEPENDENCE

An object may depend on another object in some way by including a reference to it. In such cases there is no inheritance, and the metadata of dependents and objects on which they depend have only coincidental relationship to each other. For example, when Manifestation A refers to Manifestation B by reference, A is dependent on B, and when Composite C includes Clip K, C is dependent on K.

Content Record Creation and Modification

Not all combinations of type, inheritance, and dependence are legal. The validation rules are the normative description of legal and illegal combinations. For more information, see the *Data Fields Reference.*

In order to reduce complexity, the REST API for creating and modifying objects is constrained in several ways, rather than exposing a generic data structure to be filled in. Nonetheless, all legal combinations of type and inheritance can be created and modified using the API, and all legitimate relationships to other objects can be added and removed.

The REST `Create()` call and its manifestations in the SDK use `CreationType` as an argument. This table shows the possible uses of `CreationType`.

Creation of Objects		Creation Type
Basic Objects	Referent type can be: TV, Movie, Web, Short, Supplemental	`CreateBasic`
with IsSeasonOf inheritance relationship	Only way to create Season referent type	`CreateSeason`
with information for derived type InteractiveMaterial	Only way to create InteractiveMaterial referent type	`CreateInteractive`

Creation of Objects		Creation Type
with information for derived type Series	Only way to create Series referent type	`CreateSeries`
with IsManifestationOf inheritance relationship	Only way to create an object that has IsManifestationOf relationship	`CreateManifestation`
with IsEditOf inheritance relationship	Only way to create an object that has IsEditOf relationship	`CreateEdit`
with IsClipOf inheritance relationship	Only way to create an object that has IsClipOf relationship	`CreateClip`
with IsEpisodeOf inheritance relationship	Only way to create an object that has IsEpisodeOf relationship	`CreateEpisode`
with IsCompilationOf dependent relationship	Only way to create an object of Compilation Referent Type	`CreateCompilation`
with IsCompositeOf dependent relationship	Only way to create an object of Composite Referent Type	`CreateComposite`

A Referent Type of Movie, TV, Short, Web, or Supplemental can be changed to another one of those referent types. Other Referent Types cannot be changed. For example, the Referent Type of a record can change from TV to Movie, or from Web to Supplemental, but Series cannot change to Movie nor can Season change to TV.

Here is the summary of how relationships can interact with each other.

Relationship	Type	Can co-exist with	Can be added after creation	Removable?
IsSeasonOf	Inheritance		No	No
IsEpisodeOf	Inheritance	IsCompositeOf	No	No
IsEditOf	Inheritance		No	No
IsClipOf	Inheritance		No	No
IsManifestationOf	Inheritance Dependence		No	No
IsCompilationOf	Dependence		No	No

Relationship	Type	Can co-exist with	Can be added after creation	Removable?
IsCompositeOf	Dependence	IsEpisodeOf	1) to episodes 2) to basic objects not of Composite referent type	Yes, if the referent type is not Composite
IsPromotionOf	Lightweight	Any	Yes	Multiple instances allowed on an object
IsPackagingOf	Lightweight	Any	Yes	Multiple instances allowed on an object
IsAlternateContentFor	Lightweight	Any	Yes	Multiple instances allowed on an object
IsSupplementTo	Lightweight	Any	Yes	Multiple instances allowed on an object

Alternate ID

The Alternate ID field is of particular significance in the EIDR metadata schema. It plays an important role in ensuring the interoperability of EIDR IDs with other existing ID systems.

The field consists of a type and a value. For example, an Alternate ID could have a type of ISRC and a value of FR-UM0-99-12345. Proprietary IDs are supported as well, with an added attribute giving the domain within which the ID is valid.

As another concrete example, the field could include references to a work's ISAN, allowing cross-referencing between ISAN-registered works and the full hierarchy of commercial edits, manifestations, and other records that may be registered in EIDR for the same family of records. If the work's ISAN is given as an alternate ID for an object at or near the root of an EIDR tree, it can be found from any of that object's descendants.

The Alternate ID field can also be used by metadata vendors to link EIDR records to vendor IDs that reference external sources of commercial metadata for the asset. Studios or other content producers may cross-reference to internal IDs used for other distribution or tracking purposes. EIDR serves as a useful cross-referencing tool for access to a wide variety of external sources of data about each registered asset.

Aliases and Deletion Model

IDs in DOI registries must be permanent, and the records to which they refer are intended to be permanent. The EIDR terms of use contain restrictions limiting the ability to delete a record. All records should be permanent and persistent absent special circumstances allowing aliasing or other extraordinary changes to the registry. If a record is inaccurate or otherwise corrupted, there are two ways of repairing the situation:

- The ID can be aliased to another more correct record. This is used when a duplicate is registered mistakenly, or when a changed understanding of the

underlying assets means that they should now be viewed as identical. Both IDs will exist, but both will resolve to the same underlying EIDR metadata,
- If an ID really must be deleted (because it was a complete error, such as a mistaken registration) the underlying metadata is removed and the ID is aliased to a *tombstone record*. The ID can still be resolved, which is important if it ever made its way into external systems, but the requesting entity will know the underlying object no longer exists.

An alias is a simple indirection from one DOI to another. An alias is not intended as a general tool; rather, it should only be used for correcting errors. For example, if a film mistakenly gets two IDs (perhaps because of problems involving workflow and working titles) the incorrect one can be aliased to the correct one. As a concrete example, if an incorrect ID is burned onto a BD-Live disk and is subsequently corrected by aliasing, that ID can still be resolved by the server in response to client requests for the deprecated ID. The server can treat the content as valid simply because the ID is resolvable, or add extra checking.

A Series or Season object cannot alias to a record that has any of these relationships: isEpisode, isEdit, isClip, isCompilation, isComposite. isManifestation.

Deleted objects are aliased to a *tombstone*. Because the tombstone object is of type `Restricted`, only three of its fields have guaranteed values. Other fields are not used.

Field	Value
ID	10.5240/ 0000-0000-0000-0000-0000-X
Structural Type	Restricted
Resource Name	“EIDR Tombstone Object”

The tombstone object is not indexed, so it will never be returned from queries.

Alias chains: An object may be aliased to an object that is itself an alias. If `followAlias` is `true` when doing a resolution (the normal case) the chain is followed until it ends up at a resolvable record or the chain is more than five levels deep, in which case the registry returns an `aliasContinuation`, which contains the last object reached and the object to which it is aliased. You can continue towards a real object by applying a resolve on the item to which the last object is aliased.

Record resurrection: If an EIDR record is erroneously aliased or deleted, it is possible to reclaim the ID and restore it to active use once again. This is a special

administrative procedure. Requests for restoration of an aliased or deleted ID should be carefully considered and only made when absolutely necessary.

Virtual Fields

Besides individual fields, an object contains two *virtual* string fields that combine multiple other base fields. Both of these are normalized (punctuation stripped, spaces collapsed) and tokenized using the default (English) rules. Stop-word filtering and stemming are not applied. This also applies to strings as queries on the virtual fields. For more information, see Appendix B: Text Processing and Queries.

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

In queries, the XPath expressions for 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 exact query might match token sequences that cross fields. It matches token sequences within individual fields predictably as it would on the single field.

The values of the virtual fields can be retrieved using `GetVirtualFields()`. This can be useful for debugging, but is not intended to be used for presentation to an end user.

Parties

A Party represents an entity such as a Registrant or a producing agent. A User is an individual (or an abstract thing that can be treated as an individual, such as software program performing an automated task).

All Users are associated with a Party.

Most Registry requests require authentication information for a User, and all requests allow it.

Only Parties have permissions in the system; a User has all the permissions associated with its parent Party.

A Party can be either *Active* or *Inactive*. An inactive Party may not make any modifications to the database; that is, it may not be a Registrant or a Writer, and all Users associated with it are similarly restricted.

There is a predefined Party representing EIDR Operations (10.5237/superparty). Only the EIDR operator has the ability to create or modify Parties.

A Party can have one or more of the following roles:

- **AssociatedOrg:** These bring forth the object being registered: a studio in the case of an abstract work; or an encoding house for the work in a final digital form; or even an anti-piracy vendor for registering a newly discovered illegal copy of a work. The important thing to remember about this field is that it refers to the Party that most recently *touched* the item. A party can only be used as an Associated Organization if it has AssociatedOrg in its AllowedRoles.
- **Registrant**. A Party, which registers content, may be a studio or an encoding house, but it may also be a Party doing bulk registration of back-catalogue items, or a Party acting on behalf of someone else. The User requesting the creation of a new object must be associated with the Registrant (a Party) given in the registration data.
- **MetadataAuthority**: This party may be included optionally by the registrant to indicate the entity the registrant thinks is most likely to have some authority over this object. For example, a metadata provider doing bulk registration may make this field the same as the production company for objects reasonably sure to be under the control of that entity, and leave it blank otherwise.
- **EncodingAgent**: Optional Party used when registering Manifestations.

Additionally, an entity can be registered as:

- **Reader**: If the entity is on an object's ACL, it can read objects and metadata that would otherwise be hidden. This applies only to "in development" objects.
- **Writer**: Can read and modify objects, but not create them.

For example, an encoding house that registers new Manifestations is the Registrant for the object. In the strictest sense, they could be the producer as well, but that is not mandated; in this example the rights holder could require that the producer for the Manifestation be the same as that for the parent object. The encoding house may also be the EncodingAgent for each item, unless some aspect has been subcontracted to a third party such as a specialist subtitle shop.

Permissions Model

A regular record can be created, modified, aliased, or deleted. An in-development record can also be Promoted. It is also possible to read certain types of administrative information about a record.

A Party (and its Users) can only create a new record if it has "Registrant" in the `AllowedRoles` field.

A Party (and its Users) must be on the object's ACL in order to be able to create, modify, or read a record's ACL or detailed provenance metadata. (Anyone can view any record or relationship in the Registry along with basic provenance metadata, except for In-Development records, as noted below).

Other actions are gated by ACLs on the individual records. Each regular record has an ACL for:

- **`Modify`:** Can contain Parties that are of type Registrant or Writer. Required to modify an object.
- **`Delete`**: Can contain Parties that are of type Registrant or Writer. Required to alias or delete an object.
- **`ReadACL`:** Can contain Parties that are of type Registrant, Writer, or Reader. Required to read any ACL.
- **`WriteACL`:** Can contain Parties that are of type Registrant. Required to modify any ACL.
- **`ReadProvenance`:** Can contain Parties that are of type Registrant, Writer, or Reader. Required to read the provenance metadata. Only the Registration Authority can modify a record's Provenance information.

In-Development records have two more ACLs:

- **`Promote`**: Contains Parties of type Registrant. Required to promote an In Development object to Valid.
- **`View`**: Contains Parties of any type. Required to view In Development objects or have them returned from a query.

Access Control Lists

The following table summarizes possible permissions based on the Role. For a given object, the associated permissions for a particular Party (with a Role) may be more restrictive than what is specified in the table.

Role\Permission	*View*	*Read ACL*	*Read Provenance*	*Modify*	*Delete*	*Write ACL*	*Promote*
Registrant	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Role\Permission	*View*	*Read ACL*	*Read Provenance*	*Modify*	*Delete*	*Write ACL*	*Promote*
Writer	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reader	Yes	Yes	Yes	No	No	No	No
AssociatedOrg	Yes	No	No	No	No	No	No
MetadataAuthority	Yes	No	No	No	No	No	No
EncodingAgent	Yes	No	No	No	No	No	No

If a Role is removed from a Party, any ACL that depends on the presence of that role will disallow the action, even if the Party is in the ACL. For example, if the Writer role is removed from a Party, that Party and its associated Users will not be able to modify records for which it was already on the ACLs for Modify, Delete, WriteACL, and Promote, and will not subsequently be able to be added to the ACLs for Modify, Delete, WriteACL, or Promote on any other objects.

In development records, as opposed to v*alid* records, are records that are registered but not officially released. It is not recommended that records be in this state. They have two more ACLs than valid records:

- `Promote`: Entities on this list must be of type Registrant. Entities on this list can promote an In Development object to Valid.
- `View`: Entities on this list can be of any type. Entities on this list can view an in-development object or have them returned from a query.

5 Content Read Operations

There are several ways of getting information about an EIDR ID:

- *Resolution* provides the metadata associated with a particular EIDR ID in various formats, which are:
  - Full
  - Self-defined
  - DOI Kernel
  - Inherited
- *Provenance* returns information about the ID's creation and modification. It is a special case of resolution.
- *Traversal* provides information about how the record fits into the object hierarchy, using inheritance, lightweight, and dependence relationships.
- *Queries* return records with metadata matching certain criteria.
- *Matching* returns information regarding potential duplicates for a given candidate record. Matching can be used prior to creating a new record as part of an external deduplication process to help prevent duplicate registrations from reaching the registry.

Resolution

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

Content metadata can be requested in these formats:

Full: This gives an object's complete metadata, filling in inherited fields and fully realizing any language additions or replacements.

DOIKernel: Returns metadata formatted according to the DOI Kernel metadata for a referentCreation. Please see *EIDR Data Fields Reference* for full details.

SelfDefined: Returns metadata found on the object itself, ignoring inheritance and language accumulation.

Inherited: Returns only metadata which has been inherited from an ancestor.

Simple: Returns minimal information, including the name, structural type, referent type, primary language, release date, and status of an object, along with skeletal descriptions of its relationships.

Provenance: Returns information about an object's creation and modification history.

Party and Video Service Provider metadata can be requested as Full or DOIKernel. See the *EIDR DOI Mapping* for details on the latter.

Traversals

Traversals are used for discovering how the content record fits into the graph that represents its relationship to other objects.

Traversals stop at the first object for which the requester does not have read permission. The standard Permission error is used only for permission errors on the ID in the original request.

Traversals always follow aliases.

All traversals return results in depth-first order. With traversals you can perform the following operations:

- Find ancestors
- Get series ancestry
- Get remotest ancestor
- Find descendants
- Get leaf descendants
- Get parent
- Get children
- Get dependent objects
- Get lightweight relationships.

Queries

The query facility takes a set of metadata criteria as input and returns `eidr:simpleDataInfoType` for all objects that match those criteria. The simplest query tests a single metadata field. More complex queries can be built up by grouping simple queries together with standard logical expressions.

Queries can be based on the content record's metadata, its modification history (provenance), and its virtual fields. For more information and examples of queries, see Appendix B: Text Processing and Queries.

Using Traversals and Queries to Find Records

This query finds all records with “Kung Fu” in the title:

```
(/FullMetadata/BaseObjectData/ResourceName "Kung Fu")
```

This query finds all records modified in 2013:

```
(/ProvenanceMetadata/LastModificationDate = 2013)
```

You can perform a rooted query to get the descendants of a specified object, such as the Episodes for a Series or Season, as shown in the following example using the EIDR command line tools:

To find all the Episodes registered for a Series with EIDR content ID 10.524/301C-0DFA-B184-5448-BB3E-I as the root object, create the file query-ep.txt containing

```
 (/FullMetadata/ExtraObjectMetadata/EpisodeInfo EXISTS)
```

and run

```
QueryTool -i query-ep.txt -r 10.5240/301C-0DFA-B184-5448-BB3E-I-t full
```

to see the full metadata for all the episodes of the Series.

Similarly,

```
QueryTool -i query-ep.txt -r 10.5240/301C-0DFA-B184-5448-BB3E-I -n
```

will tell you how many episodes the Series contains.

This example finds all items that are packaging of a particular record:
`(/FullMetadata/ExtraObjectMetadata/PackagingInfo/ID 10.5240/259A-5359-3425-8B6E-C169-A)`

Other lightweight relationships (such as Promotion) are analogous. You can find anything that is a packaging of something else with
`(/FullMetadata/ExtraObjectMetadata/PackagingInfo EXISTS)`

6 Using Parties

Parties are used as Production or Distribution companies in content records and as the entities to which Users are attached and give them permissions (both registry-wide, such as registration, and record-specific, such as write access to a particular record).

The public Party API allows you to retrieve information about Parties and Users in several ways:

- Search for Parties by name
- Retrieve results from an alphabetized catalog of all the Parties
- Resolve a Party (as EIDR or DOI kernel metadata)
- Resolve a User (as EIDR metadata)

There is a separate administrative API used by the Registry Operator for creating and modifying Parties.

Because there are tens of thousands of production companies, some of which existed to produce a single movie, and because many production companies are related to each other, it is often hard to get reliable information. This in turn can lead to incorrect creation of Party records – for example, "XYZZY Films" may or may not be the same entity as "XYZZY Productions", or they both could be one-time subsidiaries of "XYZZY LLC".

To correct this, EIDR has a process to deprecate duplicate or erroneous Parties. After the request to deprecate a party (containing a preferred party to be used instead) is submitted to the Registry Operator, the deprecated Party is marked Inactive.

In both cases, the names associated with the deprecated Party are added as alternate names to the preferred (correct) party and uses of the deprecated Party are replaced by the preferred Party. Applications should make sure they do not use (or allow users to use) deprecated Parties when registering or modifying content records.

7 Deduplication

A good ID is unique; an ID represents a single object, and a single object is represented by only one ID. In order to guarantee universal uniqueness, EIDR content registrations go through a central system that uses a deduplication module to guarantee that an object is unique. Once a unique ID is assigned to an object, the ID becomes a persistent and permanent part of the registry, available for use by the media and entertainment ecosystem.

Overview

The deduplication module responds to a registry request with one of four outcomes.

- *No Duplicate*: The record submitted is unique.
- *Duplicate:* The record submitted is a duplicate of an existing object in the registry.
- *Potential Duplicate:* There is a high likelihood that the record submitted is a duplicate of a record or one of many records in the registry. Each request specifies whether a potential duplicate is rejected or sent for manual review.
- *Rejected:* The submitted record was erroneous or ambiguous and could not be processed further by the registry operator. After correcting the errors or omissions, the record may be re-submitted.

When an attempt is made to register or modify an EIDR content record, the system first decides if the action would result in there being duplicate records in the database. This may occur, for example, if two back-catalogs contain records for the same movie or TV show. Allowing the registration or modification would violate the principal of one record/one ID.

There are sets of rules for determining candidates that match the newly created or modified record, from which a matching system generates scores. Different referent types and relationships have different scoring rules. Scores are computed against two thresholds:

- Low threshold: Anything below this is presumably not a potential duplicate
- High threshold: Anything greater than or equal to this is almost certainly an exact duplicate of the requested registration or modification

The normal operational mode for registration and modification is the asynchronous path. If there is only one candidate and it is above the High Threshold, the system automatically rejects the registration or modification attempt and communicates the ID of the duplicate. The item returned as the duplicate is probably good enough for the registry to use as the EIDR ID for the requested registration. However, if you are certain that the submitted record is unique, despite its apparent similarity to the identified duplicate, you may provide additional metadata sufficient to disambiguate the record and re-submit the registry request or request a manual review by setting the Operation element's dedupMode flag to "manual".

If there are no candidates above the Low Threshold, then the registration or modification is allowed to complete. Upon successful registration, you receive a new EIDR ID; otherwise, for modification requests, the underlying metadata is modified accordingly.

In all other cases having candidates above the Low Threshold for the asynchronous workflow, the attempted registration or modification is manually reviewed by EIDR to determine if it is an exact duplicate or new item registration.

In the synchronous workflow, which includes basic UI interaction, there are three possible outcomes:

- Success with no matches
- Fail with 1 high match
- Fail with multiple candidate matches.

When reviewing the results after a failure, the user takes one of the following actions:

- Identifies an exact match and uses that ID. In this case the user has completed the task, though updating the existing record with expanded metadata – including alternate IDs – is recommended practice.
- The metadata must be modified (e.g., fields added) to avoid a match: the user resubmits.
- The metadata is correct: the user submits asynchronously to trigger manual review and returns later for status from the token.
- There is a metadata or validation error: the user fixes the metadata and resubmits.

Matching API

It is possible to obtain deduplication 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 potential duplicates were found. Each item in the duplicate list includes matching scores to indicate what would happen at registration time (records above the low threshold go into manual de-duplication and a single duplicate above the high threshold should be considered a perfect match).

Tokens and Batches

The EIDR REST API has several calls, referred to as *batchable operations*, that can modify the contents of the Registry. These are `Create`, `Modify`, `AddRelationship`, `RemoveRelationship`, `ReplaceRelationship`, `Delete`, `Alias`, and `Promote`.

The EIDR API uses two approaches: single/batch requests and the immediate/async response flag.

- In reality, all EIDR requests are batch requests. What is often called a *single* or *non-batch* request is just a batch containing one request.
- The immediate/async response is a general mechanism, but can only be used for a batch size of one (a single request).

Additionally, the EIDR API uses tokens to track the status of batchable operations.

This section contains a short overview describing how batch/single requests, immediate/async responses, and tokens interact with each other, with examples of the Registry responses for the various combinations.

Note: It is important to distinguish the Response returned by the Registry from the value returned by a call through the SDK. In particular, the SDK provides some help with various errors and the registry's occasionally inconsistent error replies, but the objects returned by the SDK have a direct mapping to the Registry Response elements.

Single/Batch

All requests are submitted through the REST API as a batch. Batches of one (single requests) are treated somewhat differently from batches with multiple requests.

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

Immediate/Asynchronous

In order to guarantee uniqueness, EIDR sends requests for modifying an object's metadata to the deduplication system. In most cases this automatically returns a result. 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, giving details of the potential problems. In some cases, immediate-response requests return more detailed status information than asynchronous requests.
- If the request is not marked as immediate-response, it is sent for manual deduplication. Registry operators make a decision, which is returned to the application. This process is not real-time, and these requests are usually referred to as asynchronous.

Immediate response applies *only* to single requests, and all multiple-request batches are non-immediate. If an application requests immediate response for a batch of more than one item, the registry returns an Invalid Request error. For example:

```
<Response xmlns="http://www.eidr.org/schema" version="2.0">
 <Status>
  <Code>3</Code>
  <Type>invalid request</Type>
 </Status>
</Response>
```

Tokens

Every batchable request generates a token for the request; a multi-item request additionally generates a token for each operation in the batch. This is done with two kinds of tokens

- *Operation tokens*, which refer to individual Create, Modify, etc. requests and are returned in the `/Response/RequestStatusResults/OperationStatus/Token` XML element.
- *Batch tokens*, which refer to the status of a batch request. These are returned in the `/Response/RequestStatus/Token` element.

In addition, the user can assign a User Token to any batchable operation, generally the user's internal ID for the associated transaction. This may simplify certain integrated system workflows, since the user's system will not have to store the EIDR token.

Information is extracted from tokens with the `StatusLookup` request. Operation Tokens have detailed information about the status of an individual request (for example, a single Create or Modify). Batch tokens have information about the status of the batch and any available information about the individual items within the batch. This information includes the Operation Token and current state for each item in the batch.

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 (for example, when it is initially generated, and when it is requested via StatusLookup).

Operation Tokens

The `/RequestStatusResults/OperationStatus/Status` element will not change once it has reached a terminal state. Anything other than Pending is a terminal state. The `/RequestStatusResults/OperationStatus/Status/Code` is a numeric value from 0-5 with corresponding `OperationStatus/Status/Type` strings. These elements, as well as the codes and types for other fields, are defined in api-common.xsd.

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

For retrying error states:

- Duplicate Error should not be retried until something has changed (the metadata in the request or the metadata of the object(s) that were found as duplicates).
- Authorization Error should not be retried until something has changed (the credentials in the request, ACL of any objects involved, or the roles allowed to the requester).
- Validation Error should not be retried until something has changed (the metadata in the request or the metadata on related object(s) that caused the problem).
- Other Error is returned for various transient problems (such as bad communication with the deduplication system) and can be retried. Since it may reflect some other error, and *transient* does not necessarily mean short-lived, some caution should be used – if this error is returned a second time, it may not be productive to try it a third time.

Batch Tokens

For batches containing more than one item, once a batch has passed top-level authentication, syntax checking, etc., there are two possible states:

- 1 (batch received) means that the batch has passed the preliminary validation and is being turned into individual requests. No further information is available at this point.
- 2 (batch queued) means that the individual requests have all been submitted. In this state the individual tokens and the current state for each are returned when you call `StatusLookup` with the batch token.

Batch queued is the only terminal state for a batch token.

Scores

Scores can be returned in the response to immediate-mode requests to indicate how close any duplicate items are to the requested registration. Scores are valid ***only*** for immediate-mode requests. If they are present in the response to an async request, they should be ignored.

Polling

You must poll on a token using StatusLookup until it reaches one of the possible end states. For an Operation Token, there is really only one way to do it.

For a Batch Token you can extract all the Operation Tokens once the batch has reached the "batch queued" state 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 preferable, since you do not have 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. Using the Web UI, you can only submit one record at a time but can poll any token. If you search for a batch token you will see the operation's token results, and you can look up each operation token separately.

8 Content Create and Modify Operations

This section provides information on registration workflows, modifying records, alias operations, and delete operations.

Registration Workflows

This section contains a description of synchronous and asynchronous registration workflows, including the usage of the user interface and automated processes.

This is how people are encouraged to use the registry for new title registration (conservative for title level records, e.g. episodes in a season, new root level records such as movies, one-time-only television, new series). For Edits and Manifestations, any difference in metadata is probably distinguishing and does not require manual review (for example, a special version made for international release as opposed to domestic, MPEG4 instead of MPEG2).

In the case of someone performing a manual registration with the user interface, the process is somewhat transparent. If an asynchronous workflow requiring manual review fails with a list of potential candidates, someone using the Web UI can review the list of candidates and choose one that matches, or make their metadata distinct and proceed to manual review.

Synchronous Workflow

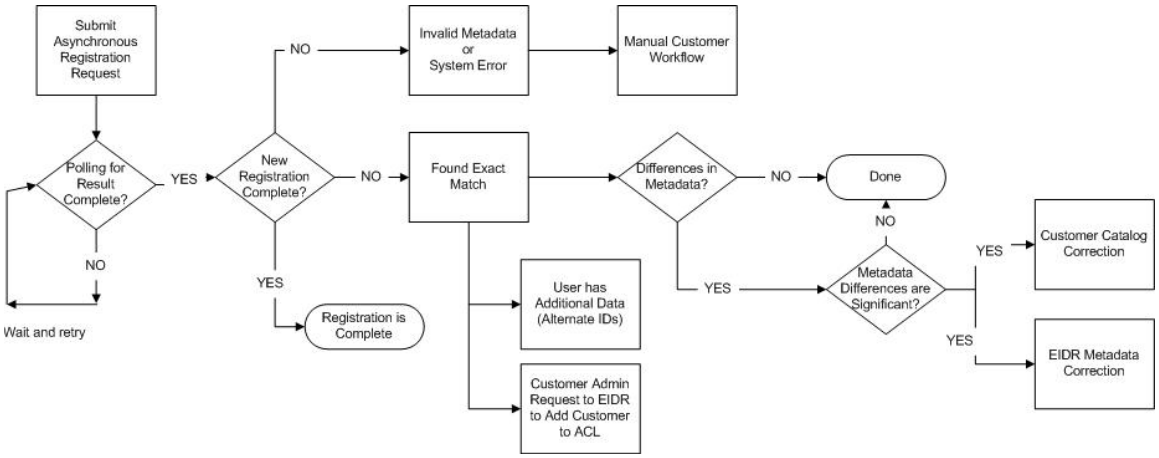
The synchronous workflow results in immediate success or immediate failure (which then requires manual review). To prevent repeated attempts by people anxious for an EIDR while accommodating automated workflows, avoid the asynchronous approach. If there are no candidates (nothing above Low Threshold), the result is immediate success with a new registration. If there is a system error, validation error, or record rejection, the user must fix the submission and resubmit. Otherwise, the system returns either a high single match or multiple candidates for consideration.

When reviewing after a failure, the user takes one of the following actions:

- Identifies an exact match. In this case the user has completed the task, though updating the existing record with expanded metadata – including alternate IDs – is recommended practice. If the user does not concur with the exact match identified, the record can be submitted for manual review by setting the dedupMode flag on the Operation element.
- The metadata must be modified: the user resubmits synchronously.
- The metadata is correct: the user submits asynchronously to trigger manual review and returns later for status from the token.
- There is a metadata or validation error: the user fixes the metadata and resubmits. There is always the chance of a system error as well, though unlikely.

API-based Asynchronous Workflow

The following diagram illustrates the asynchronous registration process using an automated API-based workflow:



An API-based application submits asynchronously and polls the result for a status of complete or incomplete. If the status is incomplete, the application waits (≈24 hours or one business day) and polls again. If the request is resolved automatically it will likely be returned within 10 seconds. It is advised to program a "back off" for slower polling.

If there is invalid metadata or a system error a manual workflow may be used, registering in the UI may be attempted, or code logic errors may be identified.

There are three possible outcomes:

- An exact match is found. Compare the metadata for differences. If there is no difference, the process is complete (done). Otherwise, determine if the differences are significant. If they are not, the process is complete (done). In such cases the EIDR user corrects their own catalog, or EIDR has incorrect information and the EIDR metadata must be fixed either through the UI or an EIDR support request.
- If the user has additional data such as an internal studio identifier (alternate IDs) to be placed on the EIDR record, the additional data are noted.
- Process modifications are needed. The EIDR user makes an administrative request to EIDR to add the User to the ACL. After the User is added to the ACL, the UI, API, or an Admin request to EIDR may be used to modify records.

In the case of new registration, there should be no corrections to the metadata and alternate IDs are part of the new registration request. The process is complete (done).

Asynchronous Workflow Tools and API

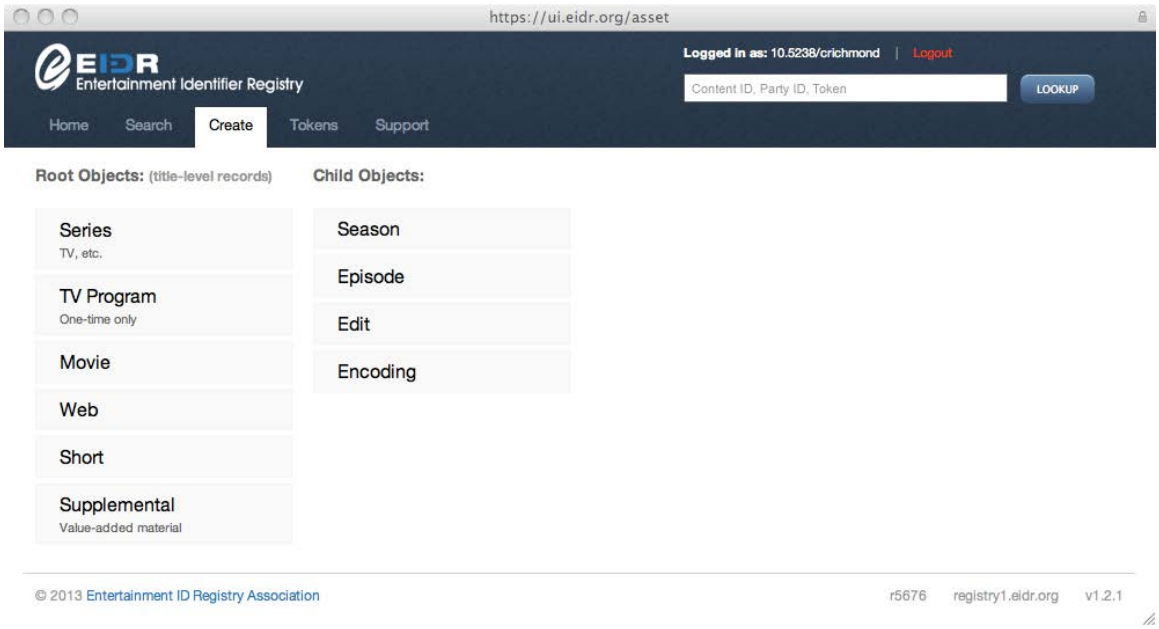
API calls can be immediate, in which case the result is returned immediately; or asynchronous, in which case the call returns a token which is used to discover the status of the request. Not all calls support asynchronous results; those that do have a flag in the interface specifying which mode to use.

User Interface Handling

The EIDR Web user interface supports common workflows for registration and modification of objects as well as search and lookup. The Web UI is built on the EIDR REST API. The following are some of the use cases supported by the UI.

- Resolve an ID: View its metadata and relationships
- Search for records based on one or more know object attributes (for example, metadata fields)
- Create objects, with special screens for common cases (such as OTO)
- Modify objects
- Add or remove relationships for a selected object
- Create objects similar to or related to an existing object in the registry
- Check status of submitted registrations.

This example shows the Create screen of the Web UI with the list of Root and Child Objects that one can register in the EIDR system:



Modifying Records

Modifying objects is done by retrieving current information on the object, modifying it, and re-submitting it, which must pass validation and deduplication. There are APIs for retrieving object metadata and modifying an object.

If the object has multiple inheritance relationships and the parent field is changed, the parent field in other inheritance relationships is changed as well.

Objects having a status of "in development" are not checked by the deduplication service. They are sent for deduplication when they are promoted to a status of "valid".

Here are the permitted ways to add relationships after object creation:

Addition of IsCompositeOf Dependent Relationship		
Basic Object	with IsEpisodeOf inheritance relationship	Base object will not be of Referent Type Composite

The lightweight relationships (IsPromotionOf, IsPackagingOf, IsAlternateContentFor, and IsSupplementTo) can be added to any object at any time.

Any relationship that can be added can be removed, with the following exceptions:

- IsCompositeOf is not removable if the object's referent type is `Composite`. This will be the case when the composite information was supplied at creation time.

Modifying objects has two components:

- Changing metadata for the type with which the object was created or could have been created.
- Changing metadata for relationships which have been added or could have been added to the object.

This table gives some examples of legal modification bases for records as they are created and modified. This table is informative, and is not exhaustive.

Initial creationType	Subsequent additions	Legal modification bases	Legal replace/remove relationship
CreateBasic	none	CreateBasic	none
CreateSeries	none	CreateBasic, CreateSeries	none
CreateClip		CreateBasic, CreateClip, CreateManifestation	

9 Error Types

The following Status Codes apply to all operations:

Status Code	Status Type	Note
0	success	Indicates that the API request succeeded.
1	system error	Should be reported to EIDR support
2	registry in read-only error	Should be reported to EIDR support unless this Registry is a mirror or is scheduled to be read-only.
3	invalid request	An API (URI) that does not exist including missing a required parameter. Remember that parameters (such as type=Simple) are case sensitive. Could also be POST data that is syntactically invalid such as missing required headers or if the end-of-line characters are not CR-LF.
4	authentication error	Invalid credentials including an Inactive account.
5	authorization error	The operation requires credentials. Or the credentials provided are not authorized to perform this operation. Check with EIDR support about this operation.
6	bad token error	There is a problem with the token ID such as one that is not syntactically valid or does not exist.
7	bad query error	There is a problem with a content record query. This could include a typographical error in an EIDR field name.
8	bad id error	There is a problem with the content ID such as one that is not syntactically valid or does not exist.

Status Code	Status Type	Note
9	syntax error	Invalid XML in a query or write operation. Examples: an incorrect namespace declaration; an element not closed; or incorrect case for an enumerated value.
10	result too long	A result was too large to fit in a REST response. This can be caused by requesting too large a page size in queries.
11	duplicate party	An Administration API error
12	duplicate user	An Administration API error
13	bad party	There is a problem with the Party ID such as it does not exist
14	bad user	There is a problem with the User ID such as it is syntactically invalid or does not exist.
15	all valid	An Administration API error
16	wrong group	An Administration API error
17	Invalid	An Administration API error
18	no parent	The object of a GetParent request is itself the root of a content record tree.
19	no children	The object of a GetChildren request is itself a leaf of a content record tree.
20	has dependents	The content record cannot be deleted because it has dependent relationships.
21	duplicate service	An Administration API error.
22	bad service	Invalid Video Service ID.
23	compatibility error	The operation cannot be supported with the requested value in the EIDR-Version header.

When the Request is a content Operation, the Response also includes an Operation Status code as described in the Operation Tokens section. The "validation error" usually includes a Details field. Examples of Details text include:

Details Text	Note
ID must be an identifier of a valid record	An operation was attempted on an invalid ID. For example to Modify an aliased record or deleting a record that is already deleted.
Must be a <Type> object	Attempt to modify an object with an incompatible data type. See the Modifying Records section.

Appendix A: DOI Proxy Parameters

EIDR resolutions can be requested from the standard DOI proxy. Resolution requests are of the form:

```
http://dx.doi.org/<Handle prefix>/<Handle suffix>
```

Object resolutions have the standard optional resolution type specifier, e.g.,

```
http://dx.doi.org/10.5240/F85A-E100-B068-5B8F-B1C8-T?locatt=type:Full
```

In the following table, "Resolution Type" is the string after the colon in the `?locatt=type:` construction. A (none) means that the type specifier is not provided.

Proxy resolutions follow alias chains, and behave like the `Resolve()` call with `followAlias` set to `true`. See `Resolve()` for information on special cases for deleted objects, over-nested aliases, etc.

EIDR item	Prefix	Resolution Type	Returns
Party	10.5237	(none)	doi:kernelMetadata with a doi:referentParty element
User	10.5238	(none)	doi:kernelMetadata with referent type of Restricted doi:referentParty element with a name of "EIDR User"
Object	10.5240	(none)	Returns metadata formatted as eidr:fullObjectInfoType.
Object	10.5240	Full	Returns metadata formatted as eidr:fullObjectInfoType.
Object	10.5240	SelfDefined	Returns metadata formatted as eidr: allSelfDefinedInfoType.
Object	10.5240	Simple	Returns metadata formatted as eidr:simpleDataInfoType
Object	10.5240	DOIKernel	Returns metadata formatted as doi:kernelMetadata.

Appendix B: Text Processing and Queries

Language-specific Filtering

There are language-specific lists for English, French, Spanish, Italian, and German for:

- Punctuation that turns into spaces
- Punctuation that collapses two words together
- Stop words that get filtered out

Language-specific rules are given in the table below. If a field has a language attribute, then the language-specific rules are applied; otherwise, English rules are applied. The query string is processed based on the language field in the queried field; if there is no language attribute, English rules are applied.

Note that the dash in language fields (e.g. de-CH) is not removed.

Language	Spacing Punctuation	Collapsing Punctuation	Stop Words
English	. , ; : ^ & ! + - = ( ) [ ] { } < > ~ # $ / * @ € £ " (double quote) - (hyphen)	' (single quote) ’ (apostrophe)	a, the, this, that, these, some, is, are, and, or, but, so, as, at, by, of, on, for, in, into, to, with, I, you, he, she, it, we, they, them, its, theirs
French	. , ; : ^ & ! + - = ( ) [ ] { } < > ~ # $ / * @ € £ " (double quote) - (hyphen) « »	' (single quote) ’ (apostrophe)	un, une, le, la, les, l, ce, ces, c, de, du, des, d, est, sont, a, ont, ne, pas, n, et, ou, mais, que, qui, qu, à, aux, sur, dans, en, par, avec, y, il, elle, ils, elles, lui, leurs, son, sa, ses, leur
Italian	. , ; : ^ & ! + - = ( ) [ ] { } < > ~ # $ / * @ € £ " (double quote) - (hyphen) « »	' (single quote) ’ (apostrophe)	ad, al, allo, ai, agli, all, agl, alla, alle, con, col, coi, da, dal, dallo, dai, dagli, dall, dagl, dalla, dalle, di, del, dello, dei, degli, dell, degl, della, delle, in, nel, nello, nei, negli, nell, negl, nella, nelle, su, sul, sullo, sui, sugli, sull, sugl, sulla, sulle, per, tra, contro, lui, lei, noi, loro, suo, sua, suoi, sue, lo, la, li, le, gli, ne, il, un, uno, una, ma, ed, se, perché, anche, come, dov, dove, che,

Language	Spacing Punctuation	Collapsing Punctuation	Stop Words
			chi, cui, non, più, quale, quanto, quanti, quanta, quante, quello, quelli, quella, quelle, questo, questi, questa, queste, si, a, c, e, i, l, o, sono, è
Spanish	. , ; : ^ & ! + - = ( ) [ ] { } < > ~ # $ / * @ € £ " (double quote) - (hyphen) « » ¿ ¡	' (single quote) ’ (apostrophe, but it is not used in Spanish)	un, unos, una, unas, el, los, la, las, este, esta, esto, estos, estas, ese, esa, eso, esos, esas, es, son, está, están, hay, y, o, pero, de, en, para, como, con, por, sobre, el, ella, ellos, ellas, se, su, sus, suyo, suya, suyos, suyas
German	. , ; : ^ & ! + - = ( ) [ ] { } < > ~ # $ / * @ € £ " (double quote) - (hyphen) „ “ « »	' (single quote) ’ (apostrophe) [Note: this means that an apostrophe at the end of a word is dropped, and one in the middle of a word for abbreviations collapses the two parts together.]	ein, einer, eine, eines, einem, einen, der, die, das, den ist, sein und, oder durch, als, von, mit, für, am, in, aus er, sie, es, sie ihn ihm, ihr, ihnen sein, siene, ihre

Field Rules

There are three kinds of matches:

- Token Match – the check is made for the existence of the tokens in any order in the candidate string. This uses the `<field> <string> <string>` syntax.
- Exact Match – the check is done for the existence of an ordered sequence of tokens in the candidate string. This uses the `<field> "<string>"` syntax.
- Complete Match – checks for the exact existence of the exact query string or not. This is generally used for fields containing controlled vocabulary or IDs,

but can also be used on some text fields. For a non-tokenized field, the query string and the field must be identical. A tokenized field must be identical to the tokenized query string. This uses the `IS` and `ISNOT` operators.

Controlled vocabulary fields have punctuation replaced by a space, and are then tokenized.

Field	Match type	Normalize	Filter Stop Words	Stem
ResourceName	Token	Yes	No	No
	Exact	Yes	No	No
AlternateResourceName	Token	Yes	No	No
	Exact	Yes	No	No
Any field called DisplayName	Token	Yes	No	No
	Exact	Yes	No	No
VirtualField (Full and self-defined) See description of Virtual Fields.	Token	Yes	No	No
	Exact	Yes	No	No
Description	Token	Yes	Yes	Yes
	Exact	Yes	No	No
HouseSequence	Complete	No	No	No
RegistrantExtra	Complete	No	No	No
RegistrantPrivate	Complete	No	No	No
AlternateID	Complete	No	No	No
FindPartiesByName	See function definition	Yes	No	No
FindPartiesFromCatalog	See function definition	Yes	No	No

Simple Queries

The metadata fields to be tested are represented with a subset of XPath notation that supports only complete paths to elements and attributes. (For more information on XPath, see http://www.w3.org/TR/xpath20/.) The XPath used in query requests can be based on:

- `/FullMetadata`, which is of type `fullObjectInfoType`: This queries across the objects' inherited metadata.
- `/ProvenanceMetadata`, which is of type `provenanceInfoType`: This queries across the objects' provenance metadata.

In these examples the parentheses are not strictly necessary, but improve legibility. Note that XML, at the protocol level, requires escaping of special characters (<, >, etc.), although procedural implementations of the API may hide that from the application.

This query finds all objects longer than 20 minutes and shorter than 40 minutes:

```
(/FullMetadata/BaseObjectData/ApproximateLength > PT20M00S) AND
(/FullMetadata/BaseObjectData/ApproximateLength < PT40M00S)
```

This finds all records modified in 2010. (See below for the precision of date comparisons.)

```
(/ProvenanceMetadata/LastModificationDate = 2010)
```

There are several kinds of simple queries, not all of which are applicable to all fields.

Exact Value: These queries use IS and ISNOT. They are applicable to

- Fields containing DOIs
- Fields containing controlled vocabulary
- Certain text fields.

Note that for a non-existent field, ISNOT returns TRUE.

Exact Value-language: This special case of Exact Value fields for language fields uses IS and ISNOT and behaves as follows:

- If only a pre-dash component is supplied in the query, it matches anything with that prefix.
- If the language code in the query has a – in it, it only matches another field that is exactly the same.
- Examples:
    - es matches objects that have es, es-ES, and es-419
    - de-CH matches de-CH, but not de or de-DE.

Order: Queries can be done using comparisons (<, <=, >=, >) as well as equality and inequality (=, <>) for fields that contain:

- Integers
- Dates
- Durations.

Existence: The existence of a field can be queried. This is useful for optional elements that represent large optional sub-blocks (e.g. the subtitle tracks of a Manifestation).

For example, this finds all objects that have information about separately encoded subtitles:

```
(/FullMetadata/ExtraObjectMetadata/ManifestationInfo/Digital/Track/
Subtitle EXISTS)
```

Text Matching:

- Text queries are case-insensitive.
- Both the text in the query string and the text stored in the registry are generally processed into tokens before matching. Tokenization consists of one or more of the following steps.
  - Normalization: Sequences of whitespace are collapsed into a single space; some punctuation is converted to spaces; and some punctuation is removed (causing concatenation of the string before it with the string after it). This gives a series of tokens.
  - Two filters can be applied to the tokens that result from normalization:
    - Stop words (small, common words, such as "the" or "in" in English or "la" and "en" in Spanish) are filtered.
    - Words are stemmed; stemming removes plurals, turns inflected words into the appropriate root, and so on.
- Strings represented using the Latin alphabet can be searched with or without diacritic significance. ASCII-based searches ignore diacritic marks ("ü" is equivalent "u"), while non-ASCII searches treat characters with different diacritics as distinct.

Search Expressions

There are two kinds of text queries:

- The form **<field> <string1>…<stringN>** is true for any field that has one or more of the strings. It is equivalent to **<field> <string1> OR <field> <string2> OR ... OR <field> <stringN>**
- The form **<field> "<string>"** is true for any field that has exactly **<string>** in it. **<string>** is tokenized before the comparison. Stated another way, the token sequence generated by **<string>** must appear exactly in **<field>.** The tokenization rules applied to applied to **<string>** are those applied to **<field>.**

The grammar for query expressions is:

```
<expression> ::= <term>
               | <expression> OR <term>
               | <expression> AND <term>
               | NOT <term>
               | ASCII (<expression>)

<term>       ::= <field> EXISTS
               | <field> <string> <string>*
```

```
| <field> "<string>"
| <field> IS "<string>"
| <field> ISNOT "<string>"
| <field> <logop> <value>
| ( <expression> )
```

Note: * is the equivalent of EBNF {} and "<term> || NOT <term>" could be EBNF "[NOT]<term>"

```
<field>      ::= legal xpath attribute
  | legal xpath element

<value>      ::= number | date | time | duration
<logop>      ::= = | <> | < | <= | > | >=
```

ASCII Searches

Using the ASCII operator in a query string changes the way Latin alphabet-based text strings are compared so that characters with and without diacritic marks are evaluated identically by mapping them all to their ASCII equivalents. The mapping is based on Unicode NFKD decomposition plus the Latin supplement (Latin-ASCII.xml) from the Unicode Common Locale Data Repository. When searching in this mode, "ü" is equivalent "u" and "ł" is equivalent to "l". This means that in most cases, ASCII versions of Latin content titles no longer need to be created manually.

To search in this mode include an ASCII modifier to one or more query Expression clauses:

```
ASCII((/FullMetadata/BaseObjectData/Credits/Actor/DisplayName Martín)
OR (/FullMetadata/BaseObjectData/Credits/Actor/DisplayName Jose))
```

This would find actors named Martin or José.

Note: ASCII searches do not automatically account for locale-dependent forms, such the "ü" in German which may be represented as "ue" in English or Latin transliterations (Romanization) of non-Latin scripts such as Cyrillic, Chinese, or Arabic, which must still be produced manually.

Notes and Examples

- The types on each side of a `<logop>` must be compatible.
- Wildcards are not currently supported; normalization and stemming cover the problems for which wildcards are generally used.
- Comparison operations for dates and times truncate to the lowest precision in the expression.
- Although ranges are not directly supported, they can be implemented using two simple queries combined by AND.

- Fields that contain controlled vocabulary are tokenized, with punctuation characters removed.
- The empty string matches nothing, rather than everything.
- For non-existent fields, all ISNOT comparisons evaluate as True. For example, if there is no CountryOfOrigin field, **`(/FullMetadata/BaseObjectData/CountryOfOrigin ISNOT fr)`** is True.
- `IS` and `ISNOT` apply to Value, Value-language, and Text fields.
    - For Value fields, they are useful for testing for controlled vocabulary words, equality of DOIs, and equality of non-tokenized fields such as HouseSequence, AlternateID, and the various private data fields.
    - For Value-language, they are used as described above.
    - For text fields, the field and the string have the same tokenization rules applied (see Appendix B: Text Processing and Queries for individual fields).
- Comparisons are done to the precision of the least precise argument. For example, a date field containing 2010 is >=, <=, and = to 10-10-2010. Using >=2012-01-01 would return the records in 2012 and later.
- Some applications may want to do queries across only metadata on the object itself, as opposed to the full metadata. This can be useful for applications whose main purpose is dealing with the metadata, rather than dealing with the objects defined by the metadata. This can be done by doing a regular query, calling `Resolve()` to return only self-defined metadata, and then examining those results.
- As an example, imagine a Registry that has objects with the following titles in the ResourceName fields:
    - Batman: The Dark Knight
    - Knight of Dark Stories
    - Dancing In The Dark
    - Darkness At Noon
    - Darkness Waits
    - The Ghost and The Darkness
    - Shanghai Knights
    - Shanghai Noon
    - Sinbad: The Battle of the Dark Knights
    - First Knight

Querying on **`/FullMetadata/BaseObjectData/ResourceName`** (abbreviated **`field`** in the table) gives these results:

Expression	Results	Notes
`field Dark`	Batman: The Dark Knight Knight of Dark Stories Dancing In The Dark Sinbad: The Battle of the Dark Knights	Anything with “Dark”.
`field "Dark Knight"`	Batman: The Dark Knight	Anything with exactly the sequence “Dark Knight”. Sinbad: The Battle of the Dark Knights is not included because titles are not stemmed.
`field Dark Knight`	Batman: The Dark Knight Knight of Dark Stories Dancing In The Dark Sinbad: The Battle of the Dark Knights First Knight	Any title that has “Dark” or “Knight”.
`field Knights`	Shanghai Knights Sinbad: The Battle of the Dark Knights	Any title with “Knights”.
`(field Dark) AND (field Knight)`	Batman: The Dark Knight Knight of Dark Stories	Any title with both “Dark” and “Knight”, in any order and any position.
`(field Dark) AND NOT (field The)`	Knight of Dark Stories	Sinbad: The Battle of the Dark Knights is not included because comparison is case-insensitive.

Note: The `ISNOT`, `NOT` and `<>` operators can be inefficient when applied globally.

Example Queries

Finding Types of Objects	
Find all Series, 1 Also works with "Season"	(/FullMetadata/BaseObjectData/ReferentType Series)
Find all Series, 2 Also works with SeasonInfo, EpisodeInfo, ClipInfo, CompilationInfo, CompositeInfo, ManifestationInfo, PackagingInfo, PromotionInfo, AlternateContentInfo, SupplementalContentInfo	(/FullMetadata/BaseObjectData/SeriesInfo EXISTS)
Find all records	(/FullMetadata EXISTS)
Find all root objects. This is done by checking for the absence of any relationship that requires a Parent.	(NOT ((/FullMetadata/ExtraObjectMetadata/SeasonInfo EXISTS) OR (/FullMetadata/ExtraObjectMetadata/EpisodeInfo EXISTS) OR (/FullMetadata/ExtraObjectMetadata/EditInfo EXISTS) OR (/FullMetadata/ExtraObjectMetadata/CompilationInfo EXISTS) OR (/FullMetadata/ExtraObjectMetadata/ClipInfo EXISTS) OR (/FullMetadata/ExtraObjectMetadata/ManifestationInfo EXISTS)))

Registrant and AssociatedOrg	
Find all "in development" records for a Registrant.	(/FullMetadata/BaseObjectData/Administrators/Registrant 10.52337/ABCD-EF01) AND (/FullMetadata/BaseObjectData/Status Dev)
Find all valid records with a particular AssociatedOrg.	(/FullMetadata/BaseObjectData/AssociatedOrg@organizationID 10.52337/ABCD-EF01) AND (/FullMetadata/BaseObjectData/Status valid)
Find all valid records with one or the other of two AssociatedOrg IDs. Generalization is left to the reader.	((/FullMetadata/BaseObjectData/AssociatedOrg@organizationID 10.52337/ABCD-EF01) OR (FullMetadata/BaseObjectData/AssociatedOrg@organizationID 10.52337/2345-6789)) AND (/FullMetadata/BaseObjectData/Status Valid)

Registrant and AssociatedOrg	
Find things registered by the EIDR Operations.	(/ProvenanceMetadata/Administrators/Registrant IS 10.5237/superparty)
Find things not registered by the EIDR Operations.	(/ProvenanceMetadata/Administrators/Registrant ISNOT 10.5237/superparty)

Looking for Possible Data Quality Problems	
All Items with English title and non-English primary language. The records may be correct (*The Hangover* was released as *Very Bad Trip* in France) but is quite often not, so it is worth investigating, especially for bulk registration.	Method 1: (/FullMetadata/BaseObjectData/ResourceName@lang en) AND (/FullMetadata/BaseObjectData/PrimaryLanguage/Language ISNOT en) Method 2: (/FullMetadata/BaseObjectData/ResourceName@lang en) AND NOT (/FullMetadata/BaseObjectData/PrimaryLanguage/Language en)
Find everything from before 1936 that is not a Movie.	(/FullMetadata/BaseObjectData/ReleaseDate <= 1936) AND (/FullMetadata/BaseObjectData/ReferentType ISNOT Movie) AND (/FullMetadata/BaseObjectData/ReferentType ISNOT Series)
Bad Season End date These can creep in if an export program uses a silly default when there is no date in the database. Change SeasonInfo to SeriesInfo for bad Series end dates. You can also generate queries like this for checking consistency for series, using either tools	(/FullMetadata/ExtraObjectMetadata/SeasonInfo/EndDate > 2014)

Looking for Possible Data Quality Problems	
and scripts or the SDK. -- Do a Full resolution -- Extract the end date -- Construct the query, using the series as the root of the query	
Find things with EIDR Operations as AssociatedOrg.	(/FullMetadata/BaseObjectData/AssociatedOrg@organizationID 10.52337/Superparty)
Statistics (The –n flag in QueryTool is useful here)	
Find all records modified since 31 December 2010. Use this to do incremental backups, setting the date to be the day before you started the last one (to avoid race conditions and time zone issues).	(/ProvenanceMetadata/LastModificationDate >= 2010-12-31)
Find anything registered in February, 2013.	(/ProvenanceMetadata/CreationDate >= 2013-02-01) AND (/ProvenanceMetadata/CreationDate < 2013-03-01)
Find all records registered by Registrant 10.5237/ABCD-EF01 in August, 2012.	(/ProvenanceMetadata/Administrators/Registrant IS 10.5237/ABCD-EF01) AND ((/ProvenanceMetadata/CreationDate >= 2012-08-01) AND (/ProvenanceMetadata/CreationDate < 2012-09-01))