

Public Domain Registry API Specification v0.9.5

I. Overview

a. General Philosophy of the Registry

The Public Domain Registry (the “Registry”) aims to be a metadata database of public domain works registered by affiliated organizations (“Registrars”). The Registry is prepared to defend against any lawsuits filed by self-proclaimed copyright holders of any of the works registered, regardless of whether the lawsuit is filed against the Registry itself, a Registrar, or a user of the Registry who has relied upon it.

The purpose of the Registry is to correct the collective action problem that has historically impeded the defense of the public domain. The public domain’s externalities have always been highly positive, but because these externalities are diffuse in nature, no single stakeholder had enough vested interest to defend (or even attempt to defend) against any threats of litigation by self-proclaimed copyright holders. By creating a Registry that handles all litigation adverse to the public domain, we aim to give the public domain a voice equal to that of the copyright domain.

b. API Technical Design Principles

In order to effectuate this Registry, there must exist the corresponding enabling technology. This API Specification (“API”) is an attempt to define the core technology used by the Registry to 1) communicate with the Registrars, including registering new works and modifying existing registrations and 2) provide registry browsing and searching capabilities to the public at large.

The design of this API follows three basic design principles: simplicity, flexibility and extensibility. Simplicity here does not mean the lack of features, but rather using the minimal amount of API to achieve maximum functionality. This is accomplished by providing relatively low level access to the database, such that unnecessary pre-processing is avoided. This freedom given the user of the API is the flexibility. Extensibility is achieved by structuring the API to allow for easy feature additions without breaking backwards compatibility or otherwise unnecessarily complicating the API.

It is our hope that this API will not only provide the basic functions necessary to implement the Registry system, but also foster a developer culture conducive to the implementation of third-party interfaces to the Registry, greatly magnifying its utility. For this purpose, read-only access is given to all members of the public. Furthermore, the modular and minimalistic nature of this API is designed to decrease implementation time not only for the Registry and Registrars, but also any third-party developers. The minimalistic nature also aims to decrease the possibility of disruptive bugs in any implementation of this API, both server-side and client-side.

II. Basic Syntax

All requests to the API where the **Scope** of the *action* is private must be in the form of a

format

The *data* field in the query must always be base64-encoded. The return value, however, is not. The following string values are possible for *format*:

1. *php* – Output of PHP's `serialize()` function.
2. *json* – JSON.
3. *wddx* – WDDX.
4. Other types may be added in the future.

action and data

The following *action* values are defined:

1. *register*
 - i) **Definition** – This action asks the Registry to register a new work.
 - ii) **Scope** – Private.
 - iii) **data** – An associative array. Each key represents a metadata field description, and the associated value contains the actual data, in string representation. The mandatory fields are:
 - a. *url* – The URL address of the entry.
 - b. *author* – The author of the work.
 - c. *workname* – The name of the work.
 - d. *copyright* – The code for copyright status. Permitted statuses include:
 - 0 – Public domain.
 - -1 – Removed/rescinded. This has no real purpose for *register* but may be useful for *modify* described below.
 - Other statuses may be added in the future (e.g. Creative Commons).
 - e. *creviewer* – The name of the person who signed off on the copyright status of the work.
 - iv) **Return Value**
 - a. Additional GRVs – 1 indicates duplication of *url*, and 2 indicates unknown copyright code. No action is taken in these cases, and RD is undefined. 3 indicates a required field is missing. In this case RD contains a list of the missing fields.
 - b. RD – A string specifying the Unique Public Domain Registry Identifier (UPDRI) of the item. An UPDRI is composed of the RIC followed by a number identifying the specific item.
2. *modify*
 - i) **Definition** – This action asks the Registry to modify or insert certain metadata field(s). Note that *modify* does not irreversibly modify the entry; rather, the history of each entry is kept in full such that *modify* changes only the most current representation of the entry.
 - ii) **Scope** – Private. In addition, a Registrar can only modify an item it created.
 - iii) **data** – An array with the following three members:
 - a. *updri* – The UPDRI of the item.
 - b. *permitinsert* – A boolean. Whether field insertion should be permitted.

- c. *newfields* – An associative array of [field name → field value]. (See the **data** defined for the *register* action, except with no required fields.)
 - iv) **Return Value**
 - a. Additional GRVs – 1 indicates invalid UPDRI, 2 indicates no permission (i.e. trying to modify an item not created by the Registrar), and 3 indicates a previously undefined field (when *permitinsert* is FALSE). No action is taken in these cases, and RD is undefined.
 - b. RD – An associative array of [field name → field value] for the UPDRI, after the modification completes.
3. *renamefield*
- i) **Definition** – This action asks the Registry to rename a field for all items of the Registrar that issued this query.
 - ii) **Scope** – Private. In addition, a Registrar can only rename fields of its own items.
 - iii) **data** – An array with the following four members:
 - a. *permitoverlap* – A boolean. Whether a field can be renamed into the same name as another preexisting field, even if the preexisting field is used by different entries in the same RIC (i.e. there is no direct replacement within an entry). If an entry already has a field with the new field name (i.e. direct replacement), setting *permitoverlap* to TRUE will cause it to be overwritten.
 - b. *oldname* – The current field name. Note that only fields associated with the RIC sending the query can be renamed (i.e. the RIC namespace is assumed).
 - c. *newname* – The new field name.
 - iv) **Return Value**
 - a. Additional GRVs – 1 indicates nonexistent *oldname*, and 2 indicates an overlapping field name (when *permitoverlap* is FALSE). No action is taken in these cases, and RD is zero.
 - b. RD – The number of entries renamed.
4. *search*
- i) **Definition** – This action asks the Registry to search for specific UPDRIs.
 - ii) **Scope** – Public.
 - iii) **data** – An array with the following two members: *type* (the type of search to be conducted) and *searchdata* (the data used by the search). The permissible values for *type* are:
 - a. *regex* – Search certain metadata fields for strings using regexes. The *searchdata* is an associative array. Each key of the array is a regex, and the value is a list of fields to search in using OR. The regexes are aggregated using AND. Regex search is always case insensitive, and all regex strings must comply with the POSIX 1003.2 standard.
 - b. *updri* – Get information on certain items, identified by UPDRI. The *searchdata* is a list of UPDRIs. If an UPDRI is invalid or does not exist, no error will be thrown, but it will not appear in the RD.
 - c. Other types may be added in the future.
 - iv) **Return Value**
 - a. Additional GRVs – 1 indicates an overly broad search. An overly broad search is a search that returns more than 1000 hits. In such a case the first 1000 hits are

returned in the RD, even though the GRV is non-zero. 2 indicates unknown *type*, and 3 indicates bad *searchdata* (e.g. invalid regex). In these cases no action is taken, and RD is undefined.

- b. RD – An associative array with the UPDRI as the key and an array of [*field name* → *field value*] with all the metadata fields for that UPDRI as the value.
5. *getfields*
- i) **Definition** – This action asks the Registry to list the names of all the fields being used.
 - ii) **Scope** – Public.
 - iii) **data** – A list of RICs to return the field usage information for, or an empty list to return the field usage information for all RICs.
 - iv) **Return Value**
 - a. Additional GRVs – 1 indicates an invalid RIC. No action is taken in this case, and RD is undefined.
 - b. RD – An associative array with each RIC as the key and an array of [*field name* → *field usage count*] as the value. The total number of entries for a particular RIC can be gotten by looking at one of the mandatory fields in *register*.
6. *getregistrars*
- i) **Definition** – This action asks the Registry to list all Registrars in the registry.
 - ii) **Scope** – Public.
 - iii) **data** – Ignored.
 - iv) **Return Value**
 - a. Additional GRVs – None.
 - b. RD – An associative array with the RIC as the key, and an associative array as the value. This second associative array has the following fields defined:
 - *name* – The name of the Registrar.
 - *url* – The home page of the Registrar.
 - Other fields may be added in the future.

V. Work Flow

An authorized Registrar is assigned a unique RIC and the corresponding password, using which the Registrar is able to register new entries and modify existing entries. The history of each entry is kept such that any modifications will not erase previous versions of the entry. The public is allowed read-only access to the API, as defined by the **Scope** of the *action*.