Cordra Technical Manual
Version: Beta 1.0.7
Date Modified: January 30th, 2016

Cordra v1.0.7 Technical Manual

Table of Contents

Introduction to Cordra

Cordra is a digital object management software that provides techniques for managing digital information. Cordra can be configured to accept metadata records that conform to one or more types. Once configured, a Cordra instance

- validates submitted records against pre-configured types,
- stores valid records for subsequent retrievals,
- indexes records for enabling queries,
- allots unique, resolvable identifiers for records for persistent identification,
- provides a RESTful API over HTTP, and, separately, a Digital Object Interface Protocol (DOIP) interface over TCP for machine-to-machine communication,
- builds a custom, schema-derived, dynamically-generated, web user interface for humans to create, retrieve, update, delete, and search records.

Release Notes

Version 1.0.7 fixes a sporadic classloading issue experienced rarely by some users.

Version 1.0.6 has several minor bugfixes: HTTPS no longer asks for a client-side certificate; Handle resolution is aware of recent GHR changes; and the internal implementation of payload indexing is streamlined.

Version 1.0.5 fixes a performance bottleneck in indexing new objects, and also includes the full source needed to build Cordra.

Version 1.0.4 adds HTTP Range requests, as well as the "indexPayloads" property to allow turning off indexing of payloads.

Version 1.0.3 changes how payloads are associated with Cordra objects. Now any Cordra object can be associated with zero or more named payloads. Payloads are no longer associated with locations in the JSON and do not need to be defined in the schema.

Version 1.0.3 improves handle minting configuration to allow handles to redirect to the Cordra UI, the JSON of the Cordra object, payloads of the Cordra object, or URLs included in the JSON. There is also a handle updater to allow changes to handle records to be performed in bulk.

Version 1.0.2 includes bug fix that prevented groups from referencing users correctly.

Technical Details

Cordra provides a RESTful HTTP API for creating, resolving, updating, deleting, and searching objects. The API allows access to payloads, metadata records, and in some cases to specific portions of those metadata records. The software also provides an automatically generated web interface for searching, viewing, and editing objects according to their type definitions.

Simply by configuring the software with schemas for the permitted types, a schema-validating API is automatically made available, and a schema-based user interface for creating and editing objects is automatically made available.

In this version, we expect submitted metadata records be represented using JSON. Each submission may optionally include payloads (e.g., datasets, video clips, or other information in digital form) that the metadata records describe.

Acceptable types can be configured using the core and validation parts of the JSON schema representation language (http://json-schema.org/latest/json-schema-core.html and http://json-schema.org/latest/json-schema-validation.html). The software takes advantage of these latest IETF drafts on JSON schema language. Those types can be additionally annotated using Cordra-specific schema attributes. The Cordra-specific attributes can be used to identify metadata properties that are handle references to other objects, provide user interface hints for better rendering, and extend user and group metadata.

The Cordra software has a built-in access control framework. As such, user and group types are pre-defined in the system, but can be extended to accept additional metadata about users and groups than what is supported by default. Each object may have an ACL listing users and groups permitted to read and/or write the object. Additional ACLs provide defaults and specify which users and groups may create objects of that type.

The Cordra software provides object-level replication to and from other Cordra instances, so that one Cordra instance may be configured to automatically copy all objects of certain types from another Cordra instance. These objects may be resolved on either instance but can only be created and edited on their originating instance. The Cordra software additionally provides a browser UI for configuration.

The Schemas section discusses the configuration aspects of the system. All other sections describe how to enable and use the RESTful API. Once configured, the system can be alternatively used using the Digital Object Interface Protocol. The protocol specification is documented here: http://hdl.handle.net/4263537/5045.

System Requirements

Minimum

JVM: Java version 8
CPU: Single core 1GHz
RAM: 500MB
HDD: 200MB

Recommended

JVM: Java version 8
CPU: Single core 2GHz
RAM: 2GB
HDD: (Storage dependent 200MB minimum)

Repositories that store large numbers of objects will need more disk space and will see better search performance with more RAM.

Schemas

Types are represented using JSON schemas. A Cordra instance may be configured with one or more schemas. The schemas are used on the server-side to ensure that objects are correctly structured, and on the client-side to automatically generate user interface for viewing and editing objects. In order to edit the schemas in your instance, open the admin interface as described in the Configuration section, and click on the "Schemas" tab.

Appendix A shows an example schema for an object that has four properties: an identifier, a name and a description. Using such a schema, the Cordra software will generate an interface for editing object that conform to that schema as shown in Figure 1.

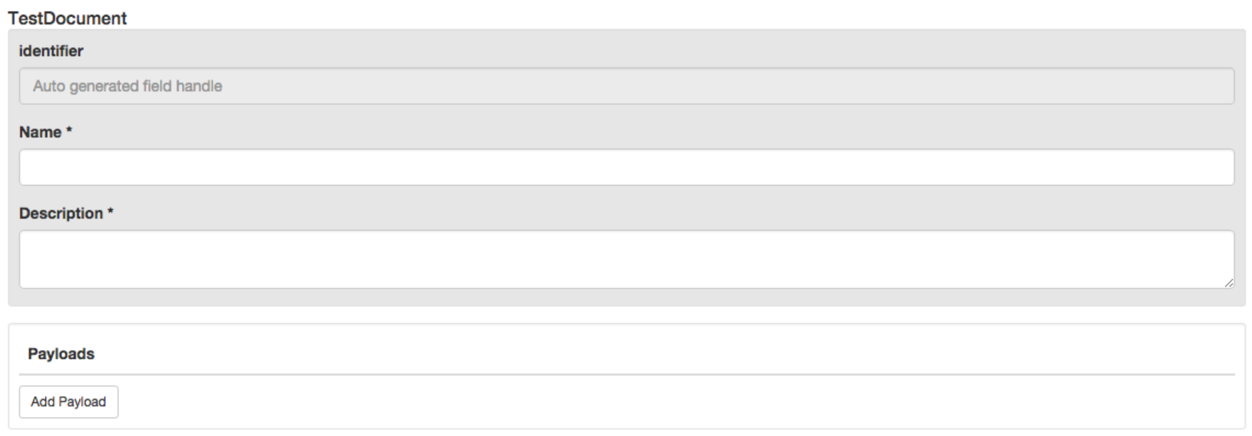

Figure 1: Edit Interface

Payloads

A Cordra object may have zero or more payloads. The presence of payloads is not specified in the schema for an object.

A payload, in this context, is information in digital form. Both the payload and its metadata record, with the help of the Cordra software, can be structured and managed as a digital object. A file is an example of digital information that can be uploaded to be managed as a payload. Along with the file, a filename and Internet media type (mime type) can also be specified. A payload is managed as a data element as part of the digital object.

When a digital object is resolved using the REST API, the full metadata record expressed in JSON is returned by default. A query parameter to the API request, "payload", allows access to payloads by name.

When an object is created or modified, the caller can optionally send a multipart request, containing a part "json" with the full metadata record of the object, and other parts with names being the payload names. Those parts specify the bytes of the payload file, plus its filename and media type. For a modification, payloads which are not modified may be omitted. Payloads can be deleted by specifying payloadToDelete params in the request.

Format keyword

The Cordra software supports validation of the "format" keyword for all values defined in the JSON Schema v4 draft specification. Additionally, other values for the "format" keyword may be used as UI hints for the automatic UI generation. Notably, arrays can have "format" : "table" to be laid out in a more compact tabular format; strings can have "format" : "textarea" to have a larger HTML textarea input box; and strings can have "format" : "password" to have a password-style hidden data entry input box.

Cordra-specific schema attributes

In addition to the usual JSON schema attributes on properties, properties can be marked with Cordra specific attributes to indicate that a property should be handled in a particular way. These are all contained within a property "net.cnri.repository".

indexPayloads

In a "net.cnri.repository" property at the top-level of a schema, "indexPayloads": false can be added to turn off indexing of payloads. This is helpful if payloads are exceptionally large and metadata is sufficient for search.

preview

These properties determine how the object should be displayed in UI previews.

preview.showInPreview

When added to a property the value of this property will be included in any UI previews of this object such as in search results or on nodes in the relationships graph.

Example:

```
"description": {
        "type": "string",
        "title": "Description",
        "net.cnri.repository": {
                "preview": {
                        "showInPreview": true
                }
        }
}
```

preview.isPrimary

The value of this property is used as the title of the object in previews. Used in conjunction with showInPreview

Example:

```
"description": {
        "type": "string",
        "title": "Description",
        "net.cnri.repository": {
                "preview": {
                        "showInPreview": true,
                        "isPrimary": true
                }
        }
}
```

preview.excludeTitle

Used in conjunction with showInPreview. Used to indicate to the UI that only the property value should be shown and not the property title.

Example:

```
"description": {
        "type": "string",
        "title": "Description",
        "net.cnri.repository": {
                "preview": {
                        "showInPreview": true,
                        "excludeTitle": true
                }
        }
}
```

type

Properties related to the typing and validation of the JSON structure.

type.suggestedVocabulary

This is similar to JSON schema enum keyword, but allows for the user to use an unspecified value. UI provides a combo box. No extra validation is performed; the suggested values are hints only.

Example:

```
"fruit": {
        "title": "Fruit",
        "type": "string",
        "net.cnri.repository": {
                "type": {
                        "suggestedVocabulary": [
                                "Apple",
                                "Banana",
                                "Orange"
                        ]
                }
        }
}
```

type.autoGeneratedField

The server will populate this field automatically with the specified type of data. Values can be "handle", "creationDate" or "modificationDate". This UI will display these fields but not allow the user to edit them. If the "handle" value is used for this property the id of the object will be auto-generated and inserted into the metadata record.

Example:

```
"identifier": {
        "type": "string",
        "net.cnri.repository": {
                "type": {
                        "autoGeneratedField": "handle"
                }
        }
}
```

type.handleReference

Properties indicating that the value in the JSON should be a handle referring to another Cordra object.

type.handleReference.types

Indicates that the property is a reference to another object. The value of this property needs to be an array of strings specifying the permitted types this object can reference. The auto generated UI for this field will provide a search input for searching for objects of the specified types.

Example:

```
"reference": {
        "title": "Reference",
        "type": "string",
        "net.cnri.repository": {
                "type": {
                        "handleReference": {
                                "types": [
                                        "type1",
                                        "type2",
                                        "type3"
                                ]
                        }
                }
        }
}
```

type.handleReference.name

Specifies a relative json path to a property that should be used to name this reference in the UI.

Example:

```
"name": {
        "type": "string",
        "title": "Name"
},
"identifier": {
        "type": "string",
        "net.cnri.repository": {
                "type": {
                        "handleReference": {
                                "types": [ "dataType" ],
                                "name": "{{../name}}"
                        }
                }
        }
}
```

In the above example "{{../name}}" indicates that the value of a sibling property should be used in the UI to name this reference. This is also used in the relationships graph to label the links.

type.handleReference.remoteRepository

Indicates that the associated property is a reference to an object stored on a remote server. The value of this property should be the URL of the remote repository that this reference should point at, ending with "/".

auth

A string property indicating a field used to specify information used in authentication and authorization.

"auth": "usersList"

Indicates that this object contains a list of users. The object can be used as a group in access control lists. The type of the associated property should be an array of strings.

Example:

```
"users": {
```

```
            "type": "array",
            "format": "table",
            "title": "Users",
            "uniqueItems": true,
            "items": {
                "type": "string",
                "title": "User",
                "net.cnri.repository": {
                    "type": {
                        "handleReference": {
                            "types": [ "user" ]
                        }
                    }
                }
            },
            "net.cnri.repository": {
                "auth": "usersList"
            }
        }
```

"auth": "username"

Indicates that this object is a user. The associated property should be a string that identifies the user. Any object that has such a username property can be used to log into the system.

Example:

```
        "username": {
            "type": "string",
            "title": "Username",
            "net.cnri.repository": {
                "auth": "username"
            }
        }
```

"auth": "password"

Used in conjunction with "username" property. Indicates that the associated property is a password. The value of this property will not be shown to the user. When the object is saved the value of this property is extracted and stored separately as a hash and salt such that the password is not stored in plain text.

Example:

```
	"password": {
		"type": "string",
		"format": "password",
		"title": "Password",
		"net.cnri.repository": {
			"auth": "password"
		}
	}
```

response.mediaType

This indicates the default Internet media type for the content of the associated property. When a client requests the URI with the jsonPointer argument for this property, and uses the text argument, then the contents of the property will be returned along with a Content-Type: header with this value.

Example:

```
	"xmlschema": {
		"type": "string",
		"format": "textarea",
		"title": "XML Schema",
		"net.cnri.repository": {
			"response": {
				"mediaType": "application/xml"
			}
		}
	}
```

Object Versioning

Cordra has support for simple object versioning. The Web UI has controls for creating a version of an object and listing existing versions of an object. When you create a version of an object a complete copy of that object is made and given a new unique identifier. The new version is marked with a timestamp for when it was versioned and the user who performed the operation. Only users with write permission for a particular object can create a version of that object.

Authentication and Access Control

A user can authenticate over the REST API using HTTP Basic Auth. A single user called "admin", which is authorized to perform most operations, is always available; the password for this user is configured in password.dct file of the Cordra web app. Other users are created as objects within the Cordra. These objects must have a type the JSON schema for which specifies properties as the username and password of that user.

The value of the password property within the JSON data is always the empty string when the object is resolved. The submitted password is hashed and salted and stored separately from the object's metadata record. The system enforces uniqueness of usernames. Since these objects are like any other object in the Cordra system, the properties associated with user objects can be changed according to the needs of any particular administrative environment; for instance, users can be associated with email addresses or names.

A sessions API is also available. For details see the REST API documentation.

Groups can also be defined as objects and can then be used to define which users have access to certain objects. A group object expressed in JSON contains an array of strings, which is the list of handles for the users in the group.

Authorization is controlled by access control lists. Each ACL is an array of strings. Those strings could be the handle identifiers of specific users or specific groups or they could be one of a set of keywords. If the ACL is left blank then only admin can perform that operation.

ACL Keyword	Description
public	Anyone can perform the operation. They do not need to be authenticated
authenticated	Any authenticated user can perform the operation.
creator	Only the creator of the object can perform the operation.
self	Only a user with the same id as the target object can perform the operation. Typically used on defaultAclWrite for user objects.

Each object has an ACL for readers and an ACL for writers. Readers can also view the ACLs; writers can also modify the ACLs. In addition to being able to specify an explicit access control list on instances of individual objects, each type can have default ACLs for objects of that type, as well as an ACL indicating who can create new objects of that type. The type-level read and write ACLs apply only to objects which do not specify their own object-level ACLs. Finally, the software can be configured with default ACLs which apply across all types which do not specify their own ACLs.

The administrative configuration APIs are authorized only for the special user "admin".

The REST API allows access to the ACLs of an object as a JSON object, with two properties "read" and "write", each an array of strings. The type-level and default ACLs are configured by specifying a JSON structure as well; an example follows.

```
{
  "schemaAcls": {
        "user": {
              "defaultAclRead": [
                    "public"
              ],
              "defaultAclWrite": [
                    "self"
              ],
              "aclCreate": []
        },
        "document": {
              "defaultAclRead": [
                    "public"
              ],
              "defaultAclWrite": [
                    "creator"
              ],
              "aclCreate": [
                    "public"
              ]
        }
  },
  "defaultAcls": {
        "defaultAclRead": [
              "public"
        ],
        "defaultAclWrite": [
              "creator"
        ],
        "aclCreate": []
  }
}
```

Configuration

The Cordra software can be configured using a web based UI. With the Cordra instance running, open a browser and go to the link:

http://<ipaddress>:<port>/admin.html

You need to be authenticated as "admin" in order to use the admin interface. Click the “Sign In” button and sign in as "admin". The default admin password is "changeit". This can be changed by editing the config.dct file or through the admin web interface.

UI Configuration

A JSON record can be used to configure some aspects of the UI. An example

```
{
  "title": "RepositoryTest",
  "allowUserToSpecifySuffixOnCreate": false,
  "initialFragment": "urls/intro.html",
  "relationshipsButtonText": "Show Relationships",
  "navBarLinks": [
          {
          "type": "query",
          "title": "All",
          "query": "*:*",
          "sortFields": "/name"
          },
          {
          "type": "typeDropdown",
          "title": "Types"
          }
  ]
}
```

Attribute name	Description
title	The text used in the title bar to identify the service.
allowUserToSpecifySuffixOnCreate	Provides a input box for the suffix of the object id when creating objects.
initialFragment	A hash fragment to be loaded if none is present when the app is loaded. This can be used to run a

	query on page load or show a document.
relationshipsButtonText	The text shown on the button used to show the relationships between objects.
navBarLinks	An array of objects used for adding links to the navigation bar. Details below.

navBarLinks

Attribute name	Description
type	Can be "query", "typeDropdown" or "url".
title	The text used on the link.
query	If the type is "query" this attribute holds the query to run.
sortFields	Sort fields for the query results.

Replication

Cordra servers can be configured to replicate from other Cordra servers. Such a configuration is defined on the server that does the pulling. An example replication configuration is shown below:

```
[
  {
        "baseUri": "http://localhost:8005/",
        "lastTimestamp": 1424371352517,
        "includeTypes": null,
        "excludeTypes": [ "test" ],
        "username": "repadmin",
        "password": "password"
  }
]
```

The replication configuration is an array of objects. One object per server being pulled from.

Attribute Name	Description
baseUri	The uri of the Cordra instance to pull from.

lastTimestamp	The timestamp that the lat pull occurred at. (read only)
includeTypes	An array of schema names to include in the replication. If null all object types will be pulled.
excludeTypes	An array of schema names to exclude from the pull. Can be null.
username	Username this server will authenticate as.
password	Password used for authenticating to the server being pulled from.

A corresponding replication credentials configuration needs to be set on the server responding to the pull requests. An example replication credentials is shown below:

```
[
 {
        "username": "repadmin",
        "password": "password"
 }
]
```

Handle Generation

If appropriately configured, Cordra will generate handles for each new object created. The handle records consist of a value redirecting the resolver to the server's API and/or user interface. Cordra needs to be configured with what base URI, ending with a slash, should be used for the URIs in the generated handle records. If the base URI is not configured, handles will not be generated.

By default handles will redirect to the Cordra UI, and allow a query parameter (locatt=view:json) to redirect to the JSON of the Cordra object. Handle records can be further configured to allow redirection to payloads or to a URL included in the JSON of the Cordra object. The configuration of handle records includes "baseUri" and "defaultLinks", where "defaultLinks" is an array of objects indicating which links will be included in the handle records. Each link has a "type" which is one of the following four options:

- "json", the JSON of the Cordra object;
- "ui", the Cordra UI for the object;
- "payload", a payload attached to the object; the link must include either "specific" indicating the payload name, or "all":true indicating that links should be generated for all payloads; or
- "url", indicating a URL embedded in the JSON of the Cordra object; the link must include an property "specific" indicating the JSON pointer to the URL.

Each link can specify "primary":true to indicate that it should be the primary redirection. Multiple links with "primary":true will result in one chosen at random when resolved by hdl.handle.net.

Non-primary links may be accessed using query parameter locatt=view:<link> where <link> is either json, ui, the name of a payload, or the JSON pointer of a URL.

An example handle minting configuration:

```
{
  "baseUri": "http://localhost:8080/",
  "defaultLinks": [
    {
      "type": "json",
      "primary": false
    },
    {
      "type": "ui",
      "primary": false
    },
    {
      "type": "payload",
      "specific": "file",
      "primary": true
    },
    {
      "type": "url",
      "specific": "/url",
      "primary": false
  ]
}
```

HTTP REST API

The Cordra software is a server-based application for creating structured objects that are referenced by Handles and conform to a set of JSON schemas specified by the administrator. Cordra exposes a RESTful HTTP API for interacting with these objects. Cordra also exposes a browser-based GUI that dynamically creates object editors, based on the JSON schemas, for creating and editing objects.

Overview

Resource	Description
`GET /objects/<id>`	Retrieves an object by id.
`POST /objects/?type=<type>`	Create an object by type.
`PUT /objects/<id>`	Update an object by id.
`DELETE /objects/<id>`	Delete an object by id.
`GET /objects/?query=<query>&pageNum=<num>&pageSize=<size>`	Search for objects.
`GET http://hdl.handle.net/<id>`	Retrieves an object via the Handle System web proxy.
`PUT /acls/<id>`	Modify the ACLs for a specific object.

Authentication

The simplest way to authenticate is to include basic auth credentials with every request. The username and password in the credentials must be with the "admin" user with the admin password or the user name of a user in the repository with the associated password.

Examples

In the following examples the schema shown below was added to the server as type "document". Multiple types can be added. The server will only accept POST and PUT requests for objects that conform to the schema corresponding to the object type; other requests will receive a 400 Bad Request response.

```
{
 "type": "object",
 "title": "Document",
 "required": [
   "name",
   "description"
 ],
 "properties": {
   "identifier": {
     "type": "string",
     "net.cnri.repository": {
       "type": {
         "autoGeneratedField": "handle"
       }
     }
   },
   "name": {
     "type": "string",
     "maxLength": 128,
     "title": "Name"
   },
   "description": {
     "type": "string",
     "title": "Description"
   },
   "creator": {
     "type": "object",
     "title": "Creator",
     "properties": {
       "fullName": {
         "type": "string",
         "title": "Full Name"
       },
       "organization": {
         "type": "string",
          "title": "Organization"
       }
     }
   }
 }
}
```

Retrieve an object by id

```
GET /objects/<id>
```

Parameters

id	required	The id of the desired object.

jsonPointer	optional	The jsonPointer into the subcomponent of the target object
payload	optional	The name of the payload to retrieve
text	optional	When present on a request which would normally result in a JSON string, the response is the contents of the JSON string
disposition	optional	For payload requests. Can be used to set the Content-Disposition header on the response; "disposition=attachment" will cause a standard web browser to perform a download operation
full	optional	If present the response is an object that contains the desired object plus metadata about that object.

Request headers

Range	optional	If present in a payload request, only retrieve the requested bytes from the payload.

Response headers

X-Schema	type of the object
X-Permission	What the calling user is authorized to do with object. READ or WRITE. Any caller with WRITE permission is also permitted to read the object.

Request

```
GET /objects/11314.5/1321d2d033b22bee1187
```

Response

```
{
  "identifier" : "11314.5/1321d2d033b22bee1187",
  "name" : "A file",
  "description" : "It's a file",
  "creator" : {
    "fullName" : "John Doe",
    "organization" : "Acme Corp."
  }
}
```

```
```

Request

```
GET /objects/11314.5/1321d2d033b22bee1187?jsonPointer=/creator
```

Response

```
{"fullName":"John Doe","organization":"Acme Corp."}
```

Request

```
GET /objects/11314.5/1321d2d033b22bee1187?jsonPointer=/description&text
```

Response

```
It's a file
```

Request

```
GET /objects/11314.5/1321d2d033b22bee1187?payload=file
```

Response

```
(Contents of the payload)
```

Create an object by type

```
POST /objects/?type=document
```

Parameters

type	required	The type of the object being created. In this case "dataType"
suffix	optional	The suffix of the handle used to identify this object. One will be generated if this option is not specified.

Request Headers

Content-Type	application/json OR multipart/form-data

Response Headers

Location	URI for accessing the created object; includes the id of the created object

POST Data

```
{
  "identifier" : "",
  "name" : "A different file",
  "description" : "This one doesn't contain a file",
  "creator" : {
    "fullName" : "Jane Doe",
    "organization" : "Acme Labs."
  }
}
```

Response

```
{
 "identifier" : "11314.5/23bdf2a62a83225a1b77",
 "name" : "A different file",
 "description" : "This one doesn't contain a file",
 "creator" : {
        "fullName" : "Jane Doe",
        "organization" : "Acme Labs"
 }
}
```

To create an object with one or more payloads, POST data of type multipart/form-data must be sent. There must be one part named "json" which is the JSON data of the object to be created. Other parts have names which are the payload names. Each payload part may also have a filename and a Content-Type which are stored as the metadata of the payload.

POST Data

```
--PART-SEPARATOR
Content-Disposition: form-data; name="json"

{
  "identifier": "",
  "name": "Really a file",
  "description": "Really a file",
  "file": ""
}
--PART-SEPARATOR
Content-Disposition: form-data; name="file"; filename="a.html"
Content-Type: text/html



--PART-SEPARATOR--
```

Update an object by id

```
PUT /objects/<id>
```

Parameters

id	required	The id of the object to update.
payloadToDelete	optional	The name of an existing payload to delete. Can be used multiple times.

Request Headers

Content-Type	application/json OR multipart/form-data

```
PUT /objects/11314.5/23bdf2a62a83225a1b77
```

POST Data

```
{
  "identifier" : "11314.5/23bdf2a62a83225a1b77",
  "name" : "A different file",
  "description" : "I've changed the description",
  "creator" : {
    "fullName" : "Jane Doe",
    "organization" : "Acme Labs."
  }
}
```

Response

```
{
  "identifier" : "11314.5/23bdf2a62a83225a1b77",
  "name" : "A different file",
  "description" : "I've changed the description",
  "creator" : {
    "fullName" : "Jane Doe",
    "organization" : "Acme Labs."
  }
}
```

When updating an object with payloads, existing payloads can be omitted from the uploaded JSON data. Those payloads will be unchanged. New payloads and modified payloads should be included as parts in a multipart/form-data request, as for the object creation API. Additionally a payload can be deleted by including its names as the value of a “payloadToDelete” parameter. Multiple “payloadToDelete” parameters is allowed.

Search for objects

```
GET /objects/?query=file&pageNum=0&pageSize=10
```

Parameters

query	required	The query to be processed.
pageNum	optional, default 0	The desired results page number. 0 is the first page..
pageSize	optional	The number of results per page. If omitted no limit.

Response

```
{
 "size": 1,
 "pageNum": 0,
 "pageSize": 10,
 "results": [
   {
     "id": "11314.5/1321d2d033b22bee1187",
     "type": "document",
     "content": {
       "identifier": "11314.5/1321d2d033b22bee1187",
       "name": "A file",
       "description": "Its a file",
       "file": "",
       "creator": {
         "fullName": "John Doe",
         "organization": "Acme Corp."
       }
     }
   }
 ]
}
```

Delete an object by id

Parameters

```
DELETE /objects/<id>
```

id	required	The id of the object to delete.

```
DELETE /objects/11314.5/23bdf2a62a83225a1b77
```

Response: empty

Delete a payload by id and name

Parameters

```
DELETE /objects/<id>?payload=<payload>
```

id	required	The id of the object containing the payload.
payload	required	The name of the payload to delete.

```
DELETE /objects/11314.5/23bdf2a62a83225a1b77?payload=file
```

Response: empty

Retrieve an object via the Handle.Net web proxy

```
GET http://hdl.handle.net/11314.5/23bdf2a62a83225a1b77?locatt=view:json
```

Parameters

locatt	optional, view:ui or view:json default view:ui	Used to specify if the redirect should respond with the json or the user interface.

Cordra will only generate handles if the "Base URI" parameter has been configured in its administrative configuration interface. That Base URI is the URI to access the Cordra server instance, ending with a slash character. Details of handle minting configuration are above.

Versioning API (Experimental)

```
GET /versions/?objectId=20.5000.181/eb3b797f9fd544fb90fb
```

Parameters

objectId	required	The id of the object you want version information on.

Response

```
[
  {
    "id": "20.5000.181/208b07aec73a36b91a1b",
    "type": "Foo",
    "versionOf": "20.5000.181/eb3b797f9fd544fb90fb",
    "publishedBy": "admin",
    "publishedOn": 1436380157539,
    "isTip": false
  },
  {
    "id": "20.5000.181/eb3b797f9fd544fb90fb",
    "type": "Foo",
    "modifiedOn": 1433957772377,
    "isTip": true
  }
]
```

```
POST /versions/?objectId=20.5000.181/eb3b797f9fd544fb90fb
```

Parameters

objectId	required	The id of the object you want to create a version of.

Response

```
{
  "id": "20.5000.181/37b4ac94ba3e14665e04",
  "type": "Foo",
  "versionOf": "20.5000.181/eb3b797f9fd544fb90fb",
  "publishedBy": "admin",
  "publishedOn": 1436380685442,
  "isTip": false
}
```

Update the ACL for a specific object

```
PUT /acls/<id>
```

Parameters

id	required	The id of the object you want to set permissions on.

```
PUT /acls/20.5000.181/37b4ac94ba3e14665e04
```

POST Data

```
{
  "read": [
    "20.5000.215/73675debcd8a436be48e"
  ],
  "write": [
    "20.5000.215/73675debcd8a436be48e"
  ]
}
```

The post data contains two arrays, read and write. These arrays should contain the ids of the users that are given the associated permission. Note that if a user is granted write permission this implicitly grants them read permission.

Response

```
{
  "read": [
    "20.5000.215/73675debcd8a436be48e"
  ],
  "write": [
    "20.5000.215/73675debcd8a436be48e"
  ]
}
```

Appendix A: Example Schema

```
{
  "type": "object",
  "title": "TestDocument",
  "required": [
    "name",
    "description"
  ],
  "properties": {
    "identifier": {
      "type": "string",
      "net.cnri.repository": {
        "type": {
          "autoGeneratedField": "handle"
        }
      }
    },
    "name": {
      "type": "string",
      "maxLength": 128,
      "title": "Name",
      "net.cnri.repository": {
        "preview": {
          "showInPreview": true,
          "isPrimary": true
        }
      }
    },
    "description": {
      "type": "string",
      "format": "textarea",
      "title": "Description",
      "net.cnri.repository": {
        "preview": {
          "showInPreview": true,
          "excludeTitle": true
        }
      }
    }
  }
}
```