1 Cordra Software ........................................... 1
  1.1 Introduction to Cordra ................................. 1
     1.1.1 Developing Prototype Applications ............... 1
     1.1.2 Scaling Applications .............................. 2
     1.1.3 Other Features .................................. 2
     1.1.4 Experimental Features ............................ 3
  1.2 Release Notes ........................................ 3
     1.2.1 What’s New in Cordra v2.0 ...................... 3
     1.2.2 Version 1.0 ...................................... 18
  1.3 System Requirements ................................. 18

2 Application Design ..................................... 20
  2.1 Introduction ......................................... 20
  2.2 Identifiers .......................................... 21
  2.3 Schemas/Types ...................................... 21
     2.3.1 Keywords and Attributes within the JSON Schema . 23
     2.3.2 Top level properties external from the JSON Schema . 36
  2.4 Design Object ....................................... 36
     2.4.1 rep0Init.json .................................. 38
  2.5 Payloads ............................................. 38
  2.6 Authentication and Access Controls ................. 38
     2.6.1 Authentication .................................. 39
     2.6.2 Authorization .................................. 41
  2.7 Lifecycle Hooks ..................................... 45
     2.7.1 Using Hooks in JavaScript....................... 50
     2.7.2 Asynchronous Lifecycle Hooks ................... 61
     2.7.3 Throwing Errors in Schema JavaScript .......... 61
     2.7.4 Cordra Modules ................................ 62
     2.7.5 Using External Modules ......................... 68
     2.7.6 JavaScript Version .............................. 69
     2.7.7 ECMAScript Modules ............................. 69
     2.7.8 Legacy JavaScript Hooks ......................... 70
     2.7.9 Examples of Hooks .............................. 70
  2.8 Type Methods ....................................... 73
     2.8.1 Instance Methods ............................... 73
2.8.2 Static Methods ................................................. 75
2.8.3 Service-level Static Methods ............................. 76
2.8.4 Allowing GET ............................................... 76
2.8.5 Direct I/O .................................................. 76
2.8.6 Request Attributes ........................................ 77
2.9 Java Hooks and Methods ..................................... 78
2.9.1 Lifecycle Hooks ............................................ 79
2.9.2 Type Methods ............................................. 81
2.9.3 Building and Deployment ................................. 82
2.10 Handle Integration ............................................. 83
2.10.1 DO-IRP Interface ......................................... 83
2.10.2 Handle Server ............................................ 83
2.10.3 Handle Server Storage ................................. 84
2.10.4 Handle Minting Configuration ......................... 84
2.11 Digital Object Versioning ................................... 86
2.12 Digital Object Hashing ....................................... 87
3 APIs 88
3.1 Introduction .................................................. 88
3.1.1 Digital Object Interface Protocol (DOIP) Interface .... 88
3.1.2 HTTP REST API ........................................ 88
3.1.3 Digital Object Identifier Resolution Protocol (DO-IRP) Interface ................. 89
3.2 DOIP and Examples ........................................... 89
3.2.1 Configuration ............................................ 89
3.2.2 Cordra Objects serialized for DOIP .................... 91
3.2.3 DOIP Java Client ........................................ 92
3.2.4 DOIP Examples ......................................... 92
3.3 DOIP API for HTTP Clients .................................. 100
3.3.1 Request format .......................................... 100
3.3.2 Response format ....................................... 102
3.3.3 Summary of Operations ............................... 103
3.3.4 Digital Object JSON Structure ...................... 104
3.3.5 Operation Details ...................................... 104
3.3.6 Design Approach ....................................... 119
3.4 HTTP REST API and Examples ............................ 123
3.4.1 Overview ............................................... 123
3.4.2 HTTP Response Codes ................................. 125
3.4.3 Authentication ......................................... 126
3.4.4 As-User ................................................ 126
3.4.5 Request Context ........................................ 126
3.4.6 API Examples ......................................... 126
3.4.7 Administrative APIs ................................. 150
3.4.8 Experimental APIs ................................... 157
3.5 Query Syntax ................................................ 159
3.5.1 queryJson .............................................. 159
3.5.2 query .................................................. 160
4 Client Libraries 167
# Extensions and Applications

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Calling Java from JavaScript</td>
<td>228</td>
</tr>
<tr>
<td>6.1.1</td>
<td>CordraHooksSupport</td>
<td>229</td>
</tr>
<tr>
<td>6.2</td>
<td>Document Repository</td>
<td>231</td>
</tr>
<tr>
<td>6.3</td>
<td>Medical Records Application</td>
<td>231</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Narrative</td>
<td>231</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Design</td>
<td>232</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Setup</td>
<td>242</td>
</tr>
<tr>
<td>6.3.4</td>
<td>Using the Application</td>
<td>243</td>
</tr>
<tr>
<td>6.4</td>
<td>Person Registry</td>
<td>247</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Narrative</td>
<td>247</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Sample Objects</td>
<td>248</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Design</td>
<td>248</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Using the Application</td>
<td>260</td>
</tr>
<tr>
<td>6.5</td>
<td>User Registration</td>
<td>266</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Steps</td>
<td>267</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Testing It Out</td>
<td>274</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Admin-created Accounts</td>
<td>275</td>
</tr>
<tr>
<td>6.5.4</td>
<td>Full example</td>
<td>277</td>
</tr>
<tr>
<td>6.5.5</td>
<td>Additional Thoughts</td>
<td>277</td>
</tr>
<tr>
<td>6.6</td>
<td>Digital Object Linking</td>
<td>278</td>
</tr>
<tr>
<td>6.7</td>
<td>OAI-PMH in Cordra</td>
<td>281</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Example OAI-PMH URLs</td>
<td>282</td>
</tr>
<tr>
<td>6.7.2</td>
<td>Design</td>
<td>282</td>
</tr>
<tr>
<td>6.7.3</td>
<td>Setup</td>
<td>283</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6.7.4 Appendix</td>
<td>287</td>
<td></td>
</tr>
<tr>
<td>6.8 Sending Emails</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>6.8.1 Installation</td>
<td>289</td>
<td></td>
</tr>
<tr>
<td>6.8.2 SMTP Configuration</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>6.8.3 Email Templates</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>6.8.4 Sending Emails</td>
<td>290</td>
<td></td>
</tr>
<tr>
<td>6.9 External Authentication Provider</td>
<td>292</td>
<td></td>
</tr>
<tr>
<td>6.9.1 Customizing the UI to use an external authentication provider</td>
<td>296</td>
<td></td>
</tr>
<tr>
<td>6.10 Collab Prototype Software</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>6.11 Neo4j as an Additional Index</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>6.11.1 Installation Instructions</td>
<td>298</td>
<td></td>
</tr>
<tr>
<td>6.11.2 Neo4j Browser Styling</td>
<td>300</td>
<td></td>
</tr>
<tr>
<td>6.11.3 Node Labels</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>6.11.4 Relationships</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>6.11.5 Custom Operations</td>
<td>301</td>
<td></td>
</tr>
<tr>
<td>6.11.6 Reindexing Objects in Neo4j</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td>6.12 Partial Replication/Aggregation</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td>6.12.1 Installation Instructions</td>
<td>302</td>
<td></td>
</tr>
<tr>
<td>6.12.2 Behavior</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>6.12.3 Custom Operations</td>
<td>306</td>
<td></td>
</tr>
<tr>
<td>7 Tools</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>7.1 Import and Export Tools</td>
<td>307</td>
<td></td>
</tr>
<tr>
<td>7.1.1 Export</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>7.1.2 Import</td>
<td>308</td>
<td></td>
</tr>
<tr>
<td>7.1.3 Hashed directory output</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>7.1.4 Limit the objects that are exported by id</td>
<td>309</td>
<td></td>
</tr>
<tr>
<td>7.1.5 Piping export to import</td>
<td>309</td>
<td></td>
</tr>
</tbody>
</table>
1.1 Introduction to Cordra

Cordra is open source software built on the digital object foundation, and offers a distinct experience to software developers who intend to build scalable infrastructures for managing digital objects with resolvable identifiers. This documentation refers to the term *digital object* interchangeably with *Cordra object* and *object*.

Cordra saves substantial development time as it comes with ready-made functionality that developers desire ranging from user authentication and access control to information validation, enrichment, storing, and indexing. Cordra provides this functionality at scale. Most of this functionality can be customized simply via configuration.

Cordra is neither a database nor an indexer. It integrates the two and provides a unified interface. By default, Cordra is configured to use the local file system of the machine it is running on along with embedded Apache Lucene for indexing. For building complex applications, such as those that require higher reliability, Cordra supports MongoDB and Amazon S3 for storage, and Elasticsearch and Apache Solr for indexing.

In some communities, it is important for consumers to be able to locate digital objects from just their identifiers. To support such cases, the identifiers that Cordra allots are compatible with the Handle System. Cordra administrators are expected to configure their Cordra identifiers with a prefix (or prefixes) issued by credentialed parties associated with the Handle System. For details, please refer to *Identifiers* and *Handle Integration*.

From a developer standpoint, Cordra software simplifies two orthogonal technical issues: Developing prototype applications and scaling those applications.

1.1.1 Developing Prototype Applications

Cordra simplifies building prototype applications for managing NoSQL information as digital objects. (The use of relational databases for storage within Cordra requires user developed extensions.)

For most prototype needs, significant software development is not needed: data models, business rules, and access control details can be provided as configuration to Cordra. Cordra stitches together the provided configuration with built-in mechanisms to enable a fully usable information management service, one that authenticates users and applies business rules in order to protect, store, and index data. Cordra also provides simple and secure DOIP and HTTP interfaces to store and retrieve...
data. The use of DOIP is recommended for applications intended to remain accessible over long
time frames, and DOIP API for HTTP Clients is available for clients using HTTP.

1.1.2 Scaling Applications

Cordra simplifies scaling a prototype application to meet most production requirements. A collection
of Cordra nodes work in unison to share user demand, while robust machinery to support concurrent
access to objects, execution of operations, and data consistency is enabled internally. Additionally,
cross-continent and cross-cloud replication can be enabled with some configuration and planning.

Overall, Cordra helps manage NoSQL information as digital objects, allots unique, resolvable iden-
tifiers, and enables APIs to interact with those digital objects at scale, while in the backend it is
responsible for ensuring data integrity and durability.

1.1.3 Other Features

Here are some other specific features of Cordra:

- Validates supplied information against one or more pre-defined JSON schemas, and stores
  them as digital objects.
- Allots unique, resolvable identifiers (called handles) to managed objects.
- Enables linking between objects, allowing a graph of objects; if desired, links can be based on
  hashes.
- Accepts JavaScript-based rules to apply at various stages of the digital object lifecycle.
- Applies declarative access control policies over managed objects.
- Enables password as well as PKI-based authentication.
- Stores structured records and payloads in the file system, or system memory, or in scalable
  storage services such as Amazon S3 and MongoDB. Support for other services may be added
  if needed.
- Indexes structured records and payloads in an embedded fashion or via scalable indexing ser-
  vices such as Elastic and Solr.
- Includes tools for backend management, such as migration from one service to another.
- Builds a schema-driven, dynamically-generated, web user interface for humans to create, re-
  trieve, update, delete, search, and protect objects.
- Provides an administrative web interface for supplying most of the configuration.
1.1.4 Experimental Features

- **Digital Object Hashing**: Once properly configured, hashes for digital objects can be automatically generated. Verification of hashes as well as immutability checks based on linked digital objects can also be made.

- **Digital Object Versioning**: Accepts a custom identifier for the version when version object creation is requested. Supports a configuration option to specify whether to include payloads when version objects are created.

For a list of possible next steps involving Cordra software, you may refer to the Cordra Design Introduction section.

1.2 Release Notes

1.2.1 What's New in Cordra v2.0

Please visit Cordra v2.0 Features for the core features offered by version 2 of the Cordra software. Below you will find a list of incremental changes made to the software.

1.2.1.1 Changes in 2.5.2 release (2023-08-29):

**Warning**: Cordra v2.5.2 is a security update.

Cordra v2.5.0 and v2.5.1 could potentially leak “internalMetadata” of objects via the 20.DOIP/Op.Relationships.Get operation or the /relationships API endpoint. The internalMetadata of User objects includes the salt and hash of the user password. The password is not stored so is never leaked, but the leaked information potentially allows offline brute force password search. This leaking only happens if the User objects are the target of handleReference properties, for example, User objects which are members of Group objects. Additionally, this leaking is prevented if the User type has an onObjectResolution hook.

Earlier versions of Cordra can allow object retrieval which bypasses the onObjectResolution hook, via the 20.DOIP/Op.Relationships.Get operation or the /relationships API endpoint. This bypassing only happens for objects which contain a handleReference link to another object. When the relationships of the target object are accessed, in earlier Cordra versions, the source object would be retrieved bypassing any onObjectResolution hook.

**Bugfixes**:


1.2.1.2 Changes in 2.5.1 release (2023-08-07):

**Warning:** Cordra v2.5.1 is a critical security update.

Cordra v2.5.1 changes the behavior of object updates which alter the type of an object. The new default is for such updates to be prevented. If you need such updates, you will need to set design flag `enableTypeChangeViaUpdate`, and carefully consider the ACLs and lifecycle hooks that will now affect such updates.

In previous versions, only the write ACL of the old type and the lifecycle hooks of the new type would affect such an update. Unless specifically prevented, this would allow users authorized to update any object, to cause the object to have a type that the users were not expected to be authorized to create. That includes types like Schema that are only intended to be created or updated by administrators, and which allow code execution via lifecycle hooks.

Going forward, the create ACL of the new type also affects the operation, which is sufficient to prevent the security issue with typical settings. Additionally, lifecycle hooks allow more fine-grained control, both at the old type and the new type. In general, if lifecycle hooks and ACLs would allow a user to delete an object and then create an object of the new type, then (when `enableTypeChangeViaUpdate` is active) the user will be able to update the object to have the new type. Since the lifecycle hook interaction may be subtle, the ability to update types is now disabled by default.

New features:

- New design flag `enableTypeChangeViaUpdate`, false by default, can be set true to allow the standard 0.DOIP/Op.Update operation to change the type of an object.
- An update which changes the type of an object accesses ACLs and lifecycle hooks as if it were a deletion (of an object of the original type) followed by a creation (of an object of the new type). In particular the write ACL of the old type and create ACL of the new type must both allow the operation; and the `beforeDelete` hook for the old type and `beforeSchemaValidation` hook of the new type both run; and the `beforeSchemaValidation` hook runs with the context attribute `isNew` set to true. The lifecycle hook context attribute `isUpdate` is set to true for the relevant hooks in case special type-update-specific logic is wanted.

Bugfixes:

- Ensure that users can call operations they are authorized to call, even if they are not authorized to retrieve the target object.
- Fix authentication backoff when it occurs with the “root” servlet instead of under `/cordra`.
- Fix issues with startup scripts when Cordra is installed in a path with space characters.
1.2.1.3 Changes in 2.5.0 release (2023-04-30):

**Warning:** As of Cordra v2.5.0, the default JavaScript engine is GraalVM JavaScript instead of Nashorn. We expect most Cordra JavaScript to work without change in the GraalVM JavaScript engine. In case your JavaScript requires Nashorn, you can configure Cordra to continue using Nashorn by adding to config.json:

```
"javascript": {
    "engine": "nashorn"
}
```

**Warning:** The DOIP API for HTTP Clients from Cordra v2.3.0 until Cordra v2.4.0 would copy request "attributes" into request "input" for GET requests. This is not done by default in Cordra v2.5.0 and later; instead the implementation of type methods can be sensitive to both attributes and input, separately. To retain the earlier behavior, you can set the design property `useLegacyAttributesAsInputForDoipGet`.

**Warning:** As of Cordra v2.5.0, the `onObjectResolution` hook applies by default to the objects returned the the caller of create or update operations. An indicator `isCreate` or `isUpdate` is available in the hook context of the `onObjectResolution` call. To retain the earlier behavior where `onObjectResolution` is not called with the result of create or update operations, you can set the design property `useLegacySkipOnObjectResolutionForCreateAndUpdate`.

New features:

- Cordra uses GraalVM JavaScript (instead of the deprecated Nashorn engine) by default; see *JavaScript Version*.
- Cordra JavaScript can use more recent JavaScript features, up to ECMAScript 2022.
- Cordra JavaScript can use ESM modules (import/export) instead of CJS modules (require/exports); see *ECMAScript Modules*.
- Java 17 support.
- Support for hooks and type methods written in Java, see *Java Hooks and Methods*.
- Service-level hooks (in "design" JavaScript or with Java annotation @CordraServiceHooks) can serve as global default lifecycle hooks used for all object types.
- More Cordra APIs now accessible via DOIP, see *DOIP API for HTTP Clients*.
- Sort order interface in Cordra UI.
- User-defined menus and query forms for Cordra UI; see *navBarLinks*.
- Support for custom authentication UI in Cordra UI; see *Customizing the UI to use an external authentication provider*.

- Cordra JSON Schema property allowing an input to be disabled in Cordra UI; see *disableInUi*.

- Added support for Elasticsearch 8 and dropped support for Elasticsearch 6. Current supported versions of Elasticsearch are 7 and 8.

- Upgrade MongoDB Java driver dependency to 4.7, compatible with MongoDB server versions from 2.6 to 6.0 inclusive.

- New module “cordra-client” for use in JavaScript hooks, allowing arbitrary calls against the host Cordra; see *Cordra Client Module*.

- The implementations of type methods can now be responsive to request “attributes” as well as “input”; see *Request Attributes*.

- Indexing and searching of date-times via index fields prefixed with *date_*, see *Querying date-time properties*.

- Request context (see *Request Context*) is now available via DOIP by setting attributes. *requestContext*.

- The *onObjectResolution* hook now applies to the results of create and update operations as returned to the caller.

- The authenticate hook can now return an arbitrary “authContext” which is visible in other hooks; see *Authenticate Hook*.

- The *createHandleValues* hook can be defined in the standard design JavaScript, instead of only on the handleMintingConfig JavaScript.

- The *generateId* hook can be defined on individual types, not just as a service-level hook.

- Allow faceted search to specify *maxBuckets*; see *Faceted Search*.

- ReindexBatch administrative API can reindex all objects from storage, or objects selected by query; see *Reindex Batch: 20.DOIP/Op.ReindexBatch*.

- QueryParams class of Java client library now has a builder() method.

- New design flag “isReadOnly”; see *Design Object*.

- New config option for MongoDB storage “useLegacyIdField”, which is true by default (and so no change is required for existing installations). This can be set to false to use “_id” instead of “id”. Note that if you have an existing MongoDB collection you would need to migrate your data before changing this option.

- New Schema top level property “indexObjects” which can be set to false to prevent indexing of objects of that type; see *indexObjects*.

- Moved “indexPayloads” property from inside the JsonSchema to a top level property of the Schema Cordra object; see *indexPayloads*.

- New query parameter “includeScore” which, if set to true, causes search results to include the “score” from the index service.
• New query parameter “includeVersions” which, if set to true, causes search results to include Cordra version objects (see Digital Object Versioning).

• Cordra webapp is included in the distribution as cordra.war instead of ROOT.war, along with a simple ROOT.war which forwards all requests to cordra.war.

• If Cordra is configured to listen on “localhost”, now the default, it will listen on both the IPv4 localhost 127.0.0.1 and the IPv6 localhost ::1.

• Cordra can be configured to listen on multiple addresses by using config.json property “listenAddresses” instead of or in addition to “listenAddress”; similarly DOIP and DO-IRP listener configuration now allow “listenAddresses”.

• It is now possible to specify a --number-of-threads with “export-tool” and “import-tool”; see Import and Export Tools.

• Improved behavior of export-tool and import-tool with very large payloads.

• Allow search of objects by ACL using search fields like acl/readers/_.

• Index fields starting with sort_ are now searchable with the default Lucene index (in addition to their existing use for sorting and faceting).

• GET /startupStatus will return a 500 error in case of startup failure for ease of monitoring.

• Internal JavaScript module 'cordraUtil' renamed to 'cordra-util', but the older name also works.

• New worked examples Neo4j as an Additional Index and Partial Replication/Aggregation.

• OpenAPI-compatible documentation for DOIP API for HTTP Clients.

Bugfixes:

• When attempting to access a non-existent object, return “Missing object” only when the authenticated user has default read permission; otherwise return an auth error. (This matches Cordra 2.3.0 behavior inadvertently changed in Cordra 2.4.0.)

• Fixed API bug that prevented DELETE with jsonPointer.

• Improve deletion of old temporary files from webapps-temp directory.

• 500 errors using the batchUpload API will be logged.

• Ensure that method ACLs on an instance override (rather than add to) default method ACLs.

1.2.1.4 Changes in 2.4.0 release (2022-02-20):

• Upgrade log4j2 dependency to 2.17.1.

• Cordra will use the more specific “Conflict” response code (409) instead of “Bad Request” (400) when an attempt is made to create a schema or user with the same name as an existing schema or user.

• New lifecycle hook “authenticate”; see Authenticate Hook.

• New lifecycle hook “customizeQueryAndParams”; see Customize Query and Params Hook.
• New lifecycle hook "beforeStorage"; see Before Storage Hook.

• New lifecycle hook "beforeSchemaValidationWithId", which can make use of a generated identifier, in contrast to "beforeSchemaValidation" which can affect a to-be-generated identifier; see Lifecycle Hooks.

• Lifecycle hooks and type methods in JavaScript can now return promises, allowing the use of asynchronous libraries.

• Documentation of As-User feature; now admits finer control via authenticate hook.

• Numeric indexing and searching of JSON numbers via index fields prefixed with num_; see Querying numeric properties.

• New POST /batchUpload API; see Batch Upload.

• Individual Schema objects can have an “authConfig” property, which takes precedence over “schemaAcls” on the “authConfig” of the design object; see Authorization.

• Individual Schema objects may have a “getAuthConfig()” static method in their JavaScript; see Get Auth Config Hook.

• S3 storage configuration allows "endpoint" and "signerOverride" to support non-standard S3-compatible services.

• S3 storage allows getting credentials implicitly from the environment.

• Elasticsearch configuration now allows multiple baseUris.

• Cordra UI screen for accessing method calls.

• New library function cordraUtil.verifyWithKey; see CordraUtil Module.

• New library function cordraUtil.getGroupsForUser; see CordraUtil Module.

• New library function cordraUtil.validateWithSchema; see CordraUtil Module.

• cordraUtil.verifyWithCordraKey now only checks signature, does not interpret JWT claims.

• New library cordra-hooks-support to facilitate writing Cordra lifecycle hooks or type methods in Java; see Calling Java from JavaScript.

• New library cordra-embedded-test-server to facilitate starting a Cordra for automated testing.

• Key simplification: a typical Cordra install will generate only a single keypair to be used for HTTPS, DOIP, handle registration, and signing.

• TokenUsingCordraClient and the JavaScript cordra-client can now be constructed with an explicit client-side idle timeout on the cache of tokens (default is 10 minutes), after which the client will assume it needs to reauthenticate instead of using a cached token.

• Possible performance improvement with MongoDB storage by removing extraneous find calls for objects without payloads.

• Warm up JSON Schema Java objects and JavaScript lifecycle hooks for possible performance gains.

• Bugfix with Tomcat sessions and auth backoff to prevent a possible infinite loop scenario with certain unfortunately timed requests.
• Bugfix for retention of payloadReaders ACL across object update.
• Bugfix to ensure that internalMetadata is not visible to schema methods.
• Workaround for rare Gson issue of illegal access for empty collections on JDK 16+.
• Allow single-instance Cordra to be configured with Kafka-based replication, which was previously only available in a distributed Cordra installation.
• Allow faceted search on aclRead and aclWrite fields.
• Include the object id in the default index field for an object.
• Bugfix to prevent an error making a faceted query for a field not yet indexed, in the Lucene backend.
• Allow Cordra to be configured to start with neither HTTP nor HTTPS port enabled.
• The Schemalnporter tool can take both --object and --name/--schema/--javascript and will sensibly combine the input.
• Make Cordra UI more useful for recovery when there is a problem with design JavaScript.
• Ensure default character encoding of UTF-8 on HTTP requests.
• New Cordra extension called Collab in prototype form. See Collab Prototype Software.

1.2.1.5 Changes in 2.3.2 release (2021-12-15):
• Upgrade log4j2 dependency to 2.16.0 to ensure Cordra is not vulnerable to the "log4shell" vulnerability CVE-2021-45046.

1.2.1.6 Changes in 2.3.1 release (2021-12-11):
• Upgrade log4j2 dependency to 2.15.0 to ensure Cordra is not vulnerable to the "log4shell" vulnerability CVE-2021-44228.
• Upgrade Netty dependency to 4.1.69.Final.
• JsonPruner (utility class in cordra-schema-util) now prunes correctly to schemas using “oneOf”.

1.2.1.7 Changes in 2.3.0 release (2021-04-01):
• Endpoint of DOIP API for HTTP clients is /doip; see DOIP API for HTTP Clients.
• Operation 0.DOIP/Op.ListOperations and the listMethods API now only list operations/methods for which the calling user has ACL-level permission.
• Aliases for service and core operation ids in DOIP; see Aliases.
• New NPM project @cnri/cordra-tool to facilitate development and testing of Cordra configuration of types, operations, and lifecycle hooks. See the README.md in the NPM package for details.

• Global static methods can be placed on the design object and can be accessed as DOIP custom operations targeting the service object; see Service-level Static Methods.

• New lifecycle hook customizeQuery allows restricting or augmenting user-supplied queries; see Customize Query Hook.

• New lifecycle hooks afterCreateOrUpdate and afterDelete; see Lifecycle Hooks.

• Lifecycle hooks for the Design object and Type objects; see Hooks for the Design Object and Type Objects.

• The convenience API endpoints /initData, /design, and /schemas now properly apply ACLs for the Design object and Type objects.

• New query parameters “facets” and “filterQueries”; see Search for objects and Faceted Search.

• Schema validation now properly validate “format”:“date” and “format”:“time”.

• The “createHandleValues” lifecycle hook allows access to the “context” parameter, which can be used to inspect “context.effectiveAcl” to determine whether an object is public.

• The “onObjectResolution” lifecycle hook can now inspect a context property “isSearch” to determine if it is being apply to a standalone object resolution or to a search result.

• New ACL “payloadReaders” allows further restricting which object readers are authorized to read payloads.

• Type methods called using GET (or POST with empty body) allow passing input via a query parameter called “params”.

• Cordra JavaScript comes with polyfills supporting many features up to ECMAScript 2017, to facilitate transpiling next-generation JavaScript source into ES5 for use in Cordra.

• Cordra schema JavaScript allows more control over the returned server response when throwing errors; see Throwing Errors in Schema JavaScript.

• The JavaScript cordra-client now throws objects which are instanceof Error for better stack traces.

1.2.1.8 Changes in 2.2.0 release (2020-10-06):

• Cordra schemas in Type objects can now reference other Cordra schemas using $ref property; see Schema References.

• Type methods and lifecycle hooks have access to Cordra object payloads; before only the JSON portion of the digital object was accessible to those methods. See Cordra.js Module.

• Type methods can now accept and return arbitrary byte streams (instead of just JSON); see Direct I/O.
• Type methods, for use with the REST API, can specify that they can be invoked using HTTP GET (instead of just POST); see *Allowing GET*.

• The onObjectResolution hook can now control access during payload retrievals and the new onPayloadResolution hook can produce bytes or override stored bytes during a payload retrieval; see *Lifecycle Hooks*.

• The JavaScript cordra-client now works with Node.js, in addition to the browser.

• A bug was fixed to ensure the createHandleValues JavaScript hook returns handle records as intended.

• A bug was fixed to stop memory a leak issue during payload creation and update.

• A bug was fixed to expose object.metadata to beforeSchemaValidation hook on update.

• The listMethods call is now available to authorized readers (instead of just writers).

• Default authorization configuration is changed for new Cordra installs. New installs by default require authentication to read any objects (other than Type/Schema objects and the design object which are public). If you prefer the old default, edit the authorization configuration via the design object or the admin interface. Existing installations after software update will continue to work as before.

• A new cordra-schema-util library is introduced with utility functions related to Jackson and schema validation, including certain utilities formerly part of cordra-client; developers can leverage this library for performing project-specific checks.

• Authentication functionality will now use the response from onObjectResolution hook applied to user objects; before only stored user objects were considered.

• Cordra distribution now includes hdl-convert-key script to facilitate converting keys among Handle, PEM, and JWK formats.

• cordraUtil.signWithKey, cordraUtil.signWithCordraKey, and cordraUtil.getCordraPublicKey utility methods are added to the embedded JavaScript library for use in Cordra lifecycle hooks and type methods to facilitate the production and checking of signatures.

• DOIP configuration allows setting serviceName and serviceDescription, which will be included in the response to the Hello request.

• The Schemalnporter tool will preserve existing JavaScript when updating a Type/Schema object; to delete existing JavaScript during an import procedure, a new –delete-javascript flag can be specified.
1.2.1.9 Changes in 2.1.0 release (2020-05-08):

- Allow use of Elasticsearch as Cordra indexer (this did not work correctly in v2.0.0).
- Fix issue with type method call locking (object passed to method is now acquired inside the lock).
- Cordra UI object editor will now retain falsy non-required properties either fetched from server or from "Edit As Json". Falsy non-required properties can be added or removed using "Edit As Json".
- Allow type methods to take input other than JSON objects (for example, strings).
- TokenUsingCordraClient, when a request fails due to a server-invalidated token, will retry with a new token.
- Add new /search HTTP API for searching; enable search requests to be sent as JSON via POST; see Search for objects.
- Enable search with "queryJson" parameter to match content of object against supplied JSON; see Search for objects.
- Enable system properties to control trusted HTTPS certificates when using HttpCordraClient; see TLS Configuration.
- Update dependencies, notably ZooKeeper, Curator, and Kafka.
- Facilitate use of TLS with ZooKeeper, Kafka, and other services; see Enabling TLS.
- Add getOrNull method to cordra-client.js. If the digital object does not exist, get method throws an exception; getOrNull method instead returns null value.
- Support new configuration options for Solr and Elasticsearch; see Configuring Indexing Backend.
- Allow limited use of ES6 (that supported by Nashorn) in JavaScript used with Cordra types. Notably "let" and "const" are usable even in Java 8; later Java versions offer partial support for arrow functions, for...of loops, and backtick-delimited strings.
- Support sending exceptionally long queries to Solr (previously Solr would by default reject queries over seven or eight thousand characters long).
- Cordra UI now incorporates dependencies that it formerly fetched from CDNs.
- Built-in handle server now uses the same listenAddress as Cordra's HTTP interfaces by default.
- Remove unexpected error.log messaging about DOIP interface when using Cordra in a servlet container.
- Fix bug preventing use of filesystem bdbje storage with multi storage configuration.
- Filesystem bdbje storage can have the path to the Cordra data directory overridden by configuration; see Configuring Storage Backend.
- Custom storage backends can automatically receive the path to the Cordra data directory as property cordraDataDir of their configuration options; see Configuring Storage Backend.
- Use "exec" in command-line scripts to ensure that only one process is spawned.
- Copy secure properties of a digital object when creating a version of that object.
- Prevent simultaneous changes to digital objects when their versions are being created.
- Make `HttpCordraClient.getNewHttpClient` and other methods available for overriding in a subclass to change HTTP client configuration.

**Warning:** Cordra Beta v2.0 versions released after 2018-08-01 include an improved JavaScript API which is incompatible with earlier released versions of Cordra Beta v2.0. If your Cordra configuration includes schema JavaScript, see [here](#) for an upgrade path.

Cordra users upgrading from early versions of the Cordra Beta v2.0, who did not use schema JavaScript (apart from the default User schema JavaScript, which will be automatically upgraded if it has not been edited), do not in general need to take any action.

Also, earlier versions of Cordra would return all results to searches with `pageSize=0`. To restore the former behavior, you can add "useLegacySearchPageSizeZeroReturnsAll":true to the Cordra **Design Object**. By default a search with `pageSize=0` returns the number of matched objects but no object content.

As of Cordra Beta v2.0 versions released after 2019-06-01, Kafka-based replication no longer includes payloads in the replication messages. If you are using replication and you need payloads to replicate there is a boolean property “includePayloadsInReplicationMessages” that can be set to true on the Design object. Note that the current implementation of replication with “includePayloadsInReplicationMessages” may require special Kafka configuration or may not be suitable when there are large payloads.

Cordra Beta v2.0 versions released after 2019-08-02 only support Elasticsearch version 6 and 7 as indexing backends. If you have an existing Elasticsearch 5 index, you'll need to upgrade and reindex.

Cordra v2.0.0 and later only support using access tokens to create HTTP REST API sessions. If your application uses the cookie-based legacySessionsApi, you will need to upgrade to use the new Token API. To restore this former behavior, you can add “useLegacySessionsApi”:true to the Cordra design object. See **Access Token API** for details on the new API.

As of Cordra v2.0.0, all authentication requests must be made over a secure HTTPS connection. To allow authentication over insecure channels, you can add “allowInsecureAuthentication”:true to the Cordra design object.

Cordra v2.0.0 uses memory sessions by default. If you have a distributed Cordra installation which uses Tomcat session replication, you will need to configure the Cordra session manager to use Tomcat-managed sessions. See **Distributed Sessions Manager**.
1.2.1.10 Changes in 2.0.0 release (2019-10-09):

- Add built-in identifier resolution; see Identifiers and Handle Integration.
- Ensure Cordra object ids are syntactically valid handles.
- Make DOIP listener active on port 9000 by default.
- Add handle values to Cordra object id records for DOIP clients to locate objects.
- Add generateId JavaScript lifecycle hook; see Generate Object Id Hook.
- Allow handleReference in schemas to refer to objects of any types, or any types except a fixed list, instead of requiring a fixed list of allowed types.
- Allow authentication only over HTTPS by default.
- New options for HTTPS configuration, in particular to allow updating certificate without restart; see Configuring HTTPS Keys.
- Change default session management to memory sessions; add separate configuration option for using Tomcat-managed sessions. See Distributed Sessions Manager.
- Add cordraUtil.js module; see CordraUtil Module.
- Prevent MongoDBStorage from storing JSON numbers not representable as MongoDB numbers.
- Make client-supplied requestContext available to JavaScript hooks; see Request Context.
- Add parameter filter to search and retrieval APIs to allow returning only parts of the objects specified by JSON pointers.
- Upgrade dependencies; support Elasticsearch 6 and 7 (but not 5).
- Add Access Token API and deprecate legacySessionsApi.
- Add script to allow easier creation of Handle key pairs.
- Support providing jar files in data/lib and sub-directories.
- New API GET /check-credentials to test authentication whether direct or token/session-based.
- Add batch files, e.g., startup and shutdown, for Windows.
- Update technical manual significantly.
1.2.1.11 Changes in 2019-06-12 beta release:

- **Digital Object Hashing**, which allows hashes of the object content to automatically be included in object metadata.

- Fix bug to ensure that any errors resulting from sessions setup (see Distributed Sessions Manager) are visible at startup.

- Prevent creating a digital object with a zero-character, i.e., empty, identifier. If the use of previous versions of Cordra resulted in digital objects with empty identifiers, you can delete them with this recovery API call: DELETE /objects/?deleteObjectWithEmptyId.

- Ensure that initial default schemas have appropriate createdOn and modifiedOn metadata.

- Versions (see Digital Object Versioning) are now immutable by default; they can be made mutable by setting a Design object flag "enableVersionEdits".

- Improved Cordra software performance.

- Fix bug that in rare cases could allow user and group changes to not be immediately visible to the portion of Cordra process that authenticates users.

- Fixes to migration from Cordra v1.

- UI fix to prevent issues with schemas containing spaces.

- Allow configuration of cookies used for Cordra sessions; see Design Object.

- Kafka-based replication no longer includes payloads in the replication messages. If you are using replication and you need payloads to replicate there is a boolean property "includePayloadsInReplicationMessages" that can be set to true on the Design object. Note that the current implementation of replication with "includePayloadsInReplicationMessages" may require special Kafka configuration or may not be suitable when there are large payloads.

- Storage modules “custom” and “multi”; see Configuring Storage Backend.

- To facilitate clients passing contextual information to the storage backend, HTTP API calls admit a query parameter “requestContext”. This will be made available to the instance of StorageChooser used by the “multi” storage module. See Multiple Storages.

1.2.1.12 Changes in 2019-04-09 beta release:

- Fix bug which prevented starting additional webapps in data/webapps.

- Add new config.json property reindexing,async; reindexing,priorityTypes no longer causes async reindexing automatically. See Reindexing.

- Improve documentation around possible issues reindexing when using types like JavaScript-Directory.
1.2.1.13 Changes in 2019-03-29 beta release:

- Ensure that sources of internal CNRI libraries are included in distribution.
- Allow minRf to be configured in Solr indexer configuration.
- Fix client tools ExportByQuery and ImportObjects which can now optionally connect to a MongoDB backend for internal metadata.
- New server-side tools “export-tool” and "import-tool" which can connect directly to Cordra storage in order to export and import objects; also "ids-by-query" to retrieve a list of ids from a running Cordra. See Import and Export Tools.
- Improve performance of reindexing under Elasticsearch.
- Make it so that components of Cordra object “metadata” are indexed under fields with names like “metadata/createdOn”, etc.
- New MongoDB storage configuration option "maxTimeMsLongRunning", which defaults to a large value, to prevent processing timeouts on slow reindexing operations.
- New HTTP API for searches which returns only object ids instead of full objects, using query parameter “&ids”.
- Fixed bug causing incorrectly sorted search results when using MongoDB storage.
- Fixed bug causing metadata “createdOn” and “modifiedOn” to differ for a newly created object.

1.2.1.14 Changes in 2019-03-09 beta release:

- Substantial changes to UI.
- Configurable session management backend; see Distributed Sessions Manager.
- User schemas can include flags to activate/deactivate users; see “auth": “accountActive”.
- Single-instance Cordra installation allows additional jar files to be made available to Cordra by placing in data/lib directory.
- A file setenv.sh next to startup will be run by startup (for ease of setting environment variables in automatic installations).
- Remove all internal dependence on objatt_ fields in the index. This allows ignoring those fields in a Solr or Elasticsearch install, if desired to save index disk space.
- Schemas can indicate that certain fields should not be stored or retrievable plain, but instead stored as a hash and salt which can be validated. Useful for secure tokens. See secureProperty.
- New API GET /startupStatus to indicate when startup has partially failed; intended to be used in situations where HTTP access to Cordra is much easier than checking logs. See Startup Status API.
- Upgrade Jetty backend in single-instance install; now supports HTTP/2 in Java 9 or later.
- /uploadObjects API now should use POST rather than PUT.
• GET /acls now only requires read permission.

1.2.1.15 Changes in 2019-01-31 beta release:

• New objectForIndexing JavaScript hook to adjust how the object is indexed; see Example: Modification of the Indexed Object.

• Required properties with schema cordra.type.autoGeneratedField were previously populated only if present with some value, even the empty string; now they are auto-generated even if missing.

• Changed default value of reindexing configuration property batchSize to 16, which allows better performance with the default "lockDuringBackgroundReindex": true.

• Fixed UI bug which prevented saving objects with missing but not required enum and boolean properties.

• In the UI, the admin schema editor now allows editing schema JavaScript.

• In the UI, added and edited schemas are now usable immediately instead of requiring a page refresh.

• MongoDB storage now allow configuration of databaseName, collectionName, and gridFsBucketName.

• Fixed bug which could cause schemas to be unknown to Cordra after a reindex in certain configurations.

1.2.1.16 Changes in 2019-01-11 beta release:

• Improvements to logging of reindexing, and speed of reindexing when using MongoDB storage.

• UI fix to prevent possible XSS in use of Toastr to show error messages.

• Configurable ACLs for schema methods; see Authorization for Type Methods.

1.2.1.17 Changes in 2018-12-06 beta release:

• Background reindexing fix to ensure objects are (by default) locked during reindexed; see Reindexing.

• New /reindexBatch API; see Reindex Batch API.

• Update documentation for /uploadObjects API.
1.2.1.18 Changes in 2018-11-27 beta release:

- General performance improvements.
- Cordra authorization now allows groups to include other groups. Note: to make use of this feature, existing deployments will need to edit the Group schema to allow the “users” property to include handles of objects of type “Group” as well as type “User”.
- Reindexing performance improvements and new configuration options; see Reindexing.
- Ensure "Content-Type: application/json; charset=UTF-8" in more responses.

1.2.2 Version 1.0

- 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 a bug fix that prevented groups from referencing users correctly.

1.3 System Requirements

Cordra software can be deployed in at least two ways: as a single instance or as a distributed system. The full extent of system requirements depends on the deployment option selected. Please refer to these two documents for learning about those requirements: Deploying Single-Instance Cordra and Deploying Cordra as a Distributed System.

The only (virtual) machine instance in the case of a single instance deployment or each instance in the case of a distributed deployment, is expected to meet the following requirements:
### System Requirements

<table>
<thead>
<tr>
<th>Minimum</th>
<th>Recommended</th>
</tr>
</thead>
<tbody>
<tr>
<td>JVM: Java version 8</td>
<td>JVM: Java version 8 or later</td>
</tr>
<tr>
<td>CPU: Single core 1GHz</td>
<td>CPU: Single core 2GHz</td>
</tr>
<tr>
<td>RAM: 500MB</td>
<td>RAM: 2GB</td>
</tr>
<tr>
<td>HDD: 200MB</td>
<td>HDD: Storage dependent</td>
</tr>
</tbody>
</table>

Storage is used for just logging system access if external backend services are employed. Otherwise, required storage space is dependent on the number and size of digital objects that are stored and indexed as well as the number of user requests. In general, and up to a certain point, more system memory will yield better search performance.
2.1 Introduction

Cordra is a highly configurable software system for managing information as digital objects. One primary way to use Cordra is to customize it to meet your feature requirements, and then directly expose it to your API users.

Cordra offers a unified programmable interface across your storage system and indexing system, and manages information logically as digital objects. Digital objects are typed, and Cordra enforces schemas on user-supplied JSON information. It applies different validation and enrichment rules at various stages of a digital object lifecycle using hooks. In addition to JSON, it stores and indexes arbitrary information in the form of payloads. It allots identifiers to digital objects that can be resolved by consumers to learn the location of digital objects wherever the Cordra instances that manage those digital objects are deployed. It authenticates users and enforces access controls via its ACL component that manages users as well as groups as digital objects. You may want to allow your potential users to self-register their accounts in Cordra.

Cordra allows users to take snapshots of digital objects. It also allows hashing that could be used for linking between digital objects, not just in a linear order, but also as graphs of objects of defined types.

If you prefer to use Cordra for managing complex applications, and the combination of provided APIs, configurations, and lifecycle hooks are not adequate for enabling desired capabilities, additional operations can be added on the server-side environment so your clients can remain simple.

The aforementioned capabilities can be configured to the most part using types, the Design object, and startup files. By default, Cordra is configured to behave as a Document Repository, but it can be customized to support complex application needs. A glimpse of its customization power is shown in this Medical Records Application. In some cases, instead of directly exposing Cordra’s APIs to your users, you may desire to deploy applications that rely on Cordra, a simple example of which is a OAI-PMH API that librarians rely on for gathering metadata from managed information.

Cordra can be deployed on a single machine or as a distributed system. In a distributed system mode, off-the-shelf load balancers can be used to redirect traffic to the cluster of Cordra nodes. Access and error logs can also be coherently managed.

Management of complex infrastructure requires tools and tutorials related to keys management, distributed sessions management, logs management, user management, administrative interface, storage import and export, and environment migration.
Cordra software is *continuously maintained* and is offered to the public in open source form.

### 2.2 Identifiers

Cordra allots identifiers to digital objects. Such identifiers are of the form *prefix/suffix* (a prefix, a slash, and a suffix). Identifiers allow digital objects to reference other digital objects, and such linking can become the basis for enforcing complex business rules.

Cordra provides a built-in Digital Object Identifier Resolution Protocol (DO-IRP) interface enabling clients to rapidly resolve digital object identifiers to their state information, or can be configured to make identifiers resolvable by registering them at external servers; see [Handle Integration](#) for details. What state information to return can be configured using [Handle Minting Configuration](#) and [Create Handle Values Hook](#). In some communities, identifiers that are resolvable via the DO-IRP are called handles. See also the [DO-IRP v3.0 specification](#).

Per DO-IRP, prefixes play a special role. If a prefix is registered with credentialed parties called MPAs, then DO-IRP clients can auto-locate the DO-IRP provider (in this case, a given Cordra instance) and in turn auto-locate the digital objects managed by that Cordra instance wherever they are in the Internet.

Cordra APIs for creating digital objects allow users to specify the identifier (including the prefix). Admins can also specify what identifiers to use when digital objects are created with the help of [Generate Object Id Hook](#). A single Cordra instance may have objects all with the same prefix, or may have objects of multiple different prefixes. If identifiers are not specified by users or admins, Cordra allots identifiers using a default prefix, which can be configured as indicated in [Handle Minting Configuration](#). The default prefix (and any other prefixes used) may be registered with MPAs.

### 2.3 Schemas/Types

Cordra manages information as digital objects. All digital objects must be typed.

Special digital objects called Types can be used to define the structure of the rest of the Cordra objects. Rules to validate and enrich objects during various stages of a digital object lifecycle can also be specified in those Types.

Essentially, Types in Cordra enable you to express:

- The structure of your information using JSON schemas, and
- JavaScript rules to apply during various stages.

Cordra is designed to support multiple types of digital objects (additionally allowing links between them).

In spite of the broader scope of these Type digital objects, for historical reasons, the Cordra type of the Type digital object is called *Schema*.

This section discusses the schema aspect of the Type objects. Refer to [Lifecycle Hooks](#) for defining rules that could be applied during various stages of a Cordra object lifecycle.
JSON schemas within those type objects should be defined 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 schemas can additionally leverage Cordra-specific functionality via attributes as defined below. The schemas are used on the server-side to ensure that objects are correctly structured, and on the client-side to automatically generate a user interface for viewing and editing objects. In order to edit the schemas in your instance via the UI, sign in as an administrator and click on the “Admin” dropdown, then “Types”.

The following is a schema for digital objects that correspond to an example Type called Document. In this example, the Document schema defines two properties that are user-managed: a name and a description, which will be part of each digital object of that Type. Each digital object will be allotted an identifier, but that identifier can also be configured to be included in the user-managed portion of the digital object, by including it as a third property as shown in the example schema below with the property name id:

```json
{
    "type": "object",
    "title": "Document",
    "required": [
        "name",
        "description"
    ],
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "type": {
                    "autoGeneratedField": "handle"
                }
            }
        },
        "name": {
            "type": "string",
            "maxLength": 128,
            "title": "Name",
            "cordra": {
                "preview": {
                    "showInPreview": true,
                    "isPrimary": true
                }
            }
        },
        "description": {
            "type": "string",
            "format": "textarea",
            "title": "Description",
            "cordra": {
                "preview": {
                ...

(continues on next page)
The Cordra software will generate an interface for editing objects based on the schema. The interface that conforms to the above schema is shown in Figure 1.

Figure 1: Schema Editing Interface

### 2.3.1 Keywords and Attributes within the JSON Schema

#### 2.3.1.1 Schema References

Cordra supports the $ref keyword for referencing within a schema or for referencing other schemas in Cordra.

JSON Pointer fragments can be used to refer within a schema. Example:

```json
{
  "type": "object",
  "required": [ "name", "address" ],
  "properties": {
    "name": { "type": "string" },
    "address": { "$ref": "#/definitions/address" }
  },
  "definitions": {
    "address": {
      "type": "object",
      "properties": {
```
References to other Cordra schemas can also be made. By default the name of the schema can be used as the relative URI for the $ref. A JSON Pointer fragment can be included. Example:

```json
{
    "type": "object",
    "required": [ "name", "address" ],
    "properties": {
        "name": { "type": "string" },
        "address": {
            "allOf": [
                { "properties": {
                    "type": {
                        "type": "string",
                        "enum": [ "residential", "business" ]
                    }
                },
                { "$ref": "Person#/definitions/address" }
            ],
            "required": [ "type" ]
        },
        "$ref": "Person#/definitions/address"
    ]
}
```

By default, each schema is considered to have a base URI of /cordra/schemas/<Type>.schema.json, where <Type> is the name of the type. (The .schema.json extension can be omitted when referring to the schema using $ref; the file: URI scheme can be included or omitted. Relative URI resolution allows using simply “Person” to refer to the schema with name Person, and so forth.)

It is also possible to set a custom base URI for a Cordra schema. Relative URIs in $ref values in a schema are resolved with respect to its base URI, and a schema's base URI can also be used in the $ref values of other schemas to refer to it. Cordra will set a custom base URI based on the following, in this preference order:

1. A baseUri property of the Cordra Type object (that is, a sibling property to name, schema, and javascript).
2. An $id property of the JSON schema.
3. An id property of the JSON schema.

Custom base URIs facilitate using schemas in Cordra that already have URIs and relative references based on a filesystem or website.

**Current Limitations of $ref**

1. Cordra only supports JSON Pointer fragments in $ref. “Plain name” fragments are not supported.
2. Cordra does not support using $id or id to identify sub-schemas; these are only meaningful to Cordra on the root of a schema.
3. Cordra UI does not in general support displaying a form when there is recursive usage of $ref.

### 2.3.1.2 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.

### 2.3.1.3 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 cordra.

**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:**

```json
"description": {
   "type": "string",
   "title": "Description",
   "cordra": {
```

(continues on next page)
preview.isPrimary

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

Example:

```json
"description": {
  "type": "string",
  "title": "Description",
  "cordra": {
    "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:

```json
"description": {
  "type": "string",
  "title": "Description",
  "cordra": {
    "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:

```json
"fruit": {
    "title": "Fruit",
    "type": "string",
    "cordra": {
        "type": {
            "suggestedVocabulary": [
                "Apple",
                "Banana",
                "Orange"
            ]
        }
    }
}
```

**type.autoGeneratedField**

If this is a string, it is equivalent to `type.autoGeneratedField.type`.

**type.autoGeneratedField.type**

The server will populate this field automatically with the specified type of data. Values can be:

- handle
- `createdOn` (or `creationDate`)
- `modifiedOn` (or `modificationDate`)
- createdBy
- modifiedBy

The 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 inserted into the metadata record.

Example:
type.autoGeneratedField.prepend

Indicates the property is only a textual suffix of the auto-generated data. The full data is the concatenation of prepend with the value which will be stored in the property. This is typically used with type.autoGeneratedField.type = "handle". Here prepend is text which could contain more or less than a handle prefix; if a handle prefix is intended, it should end with a slash. For example, if the desired handle corresponding to value xyz is 20.5555/xyz, the schema should have "prepend" = "20.5555/".

type.autoGeneratedField.prependHandleMintingConfigPrefix

If true, behaves like prepend set to Cordra's configured handle-minting prefix followed by a slash.

type.handleReference

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

type.handleReference.types

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. Omitting the types property or setting it to an empty array indicates that this reference can point to an object of any type.

Example:

```
"reference": {
  "title": "Reference",
  "type": "string",
  "cordra": {
    "type": {
      "handleReference": {
        "types": [ ]
      }
    }
  }
}
```
type.handleReference.excludeTypes

The value of this property needs to be an array of strings specifying the types this object may not reference. If this property is used, the type.handleReference.types property should be omitted. This property creates a "every type except these excluded types" semantic.

Example:

```
"reference": {
  "title": "Reference",
  "type": "string",
  "cordra": {
    "type": {
      "handleReference": {
        "excludeTypes": ["Foo", "Bar"]
      }
    }
  }
}
```

In the above example the reference can point at an object of any type except Foo or Bar.

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"
},
"id": {
  "type": "string",
  "cordra": {
    "type": {
```
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.prepend**

Indicates the property is only a textual suffix of the handle of the referenced object. The full handle is the concatenation of prepend with the value of the property. Here prepend is text which could contain more or less than a handle prefix; if a handle prefix is intended, it should end with a slash. For example, if the desired handle corresponding to value xyz is 20.5555/xyz, the schema should have "prepend" = "20.5555/".

**type.handleReference.prependHandleMintingConfigPrefix**

If true, behaves like prepend set to Cordra's configured handle-minting prefix followed by a slash.

**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.

In order to indicate that the elements of the list are suffixes of the full user ids, it is possible to use an expanded form:

"auth": { "type": "usersList", "prepend": "..." }

or:

"auth": { "type": "usersList", "prependHandleMintingConfigPrefix": true }

Example:
If there are multiple properties with the "auth": "usersList" nested property, only the last of such properties is considered by Cordra for the user list.

"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. Cordra ensures uniqueness of usernames.

Example:

"username": {
  "type": "string",
  "title": "Username",
  "cordra": {
    "auth": "username"
  }
}
“auth”: “password”

Used in conjunction with “username” property. Indicates that the associated property is a password. The following steps are taken to ensure the security of the provided password:

The password is not directly stored by the server.

When the object is saved, the password is salted with 16 bytes generated by java.security.SecureRandom, and the result is hashed using the PBKDF2WithHmacSHA1 algorithm for 10K iterations. The final hash is then stored.

Both the hash and the salt are stored in the metadata for this object. This metadata is not part of the content of the object and cannot be retrieved using the APIs.

When a user attempts to authenticate, the password supplied with the authentication request is salted with the stored salt and hashed the same number of times as before with the same algorithm. The resulting hash is compared with the stored hash. If the two hashes match, authentication is deemed successful.

Example:

```
"password": {
   "type": "string",
   "format": "password",
   "title": "Password",
   "cordra": {
      "auth": "password"
   }
}
```

“auth”: “requirePasswordChange”

Used in conjunction with “username” and “password” on a boolean property. When that property is set to true, the user must change their password (see Password Change API) before performing any other API call. Password changes will automatically set the property to false.

Example:

```
"requirePasswordChange": {
   "type": "boolean",
   "title": "Require Password Change",
   "description": "If true a new password must be set on next authentication.",
   "cordra": {
      "auth": "requirePasswordChange"
   }
}
```
“auth”: “accountActive”

Used in conjunction with “username” and “password” on a boolean property. When that property is set to true, the user account is active and can be used to log in and perform functions. If false, the user account is inactive and will be blocked. User accounts with no accountActive property are treated as active.

Example:

```json
"accountActive": {
    "type": "boolean",
    "title": "Account active",
    "cordra": {
        "auth": "accountActive"
    }
}
```

“auth”: “publicKey”

Indicates that a public key is stored in the Cordra object for use with key-pair-based authentication (Authentication via Keys). The key should be stored in JSON Web Key (JWK) format.

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:

```json
"xmlschema": {
    "type": "string",
    "format": "textarea",
    "title": "XML Schema",
    "cordra": {
        "response": {
            "mediaType": "application/xml"
        }
    }
}
```
search.altFieldName

Indicates an alternate field name to be used in searches, in addition to the default field name of the JSON Pointer.

Example:

```
"abstract": {
  "type": "string",
  "cordra": {
    "search": {
      "altFieldName": "abstract"
    }
  }
}
```

search.altPayloadFieldNames

On the root of the schema, this indicates a mapping from payload names to alternate field names to be used in searches, in addition to the default field name of the payload name.

Example:

```
{
  ...
  "cordra": {
    "search": {
      "altPayloadFieldNames": {
        "file": "dataset",
        "documentation": "doc"
      }
    }
  }
}
```

referrable

Used to indicate that the object contains information that can be referred to. Currently this is used for JavaScript modules to be used with require or import; see Using External Modules. With "id": true and "payloads": "scripts" this indicates that payloads on the object can be referred to as JavaScript modules, with the field being the directory name and the payload name being the filename.

Example:
secureProperty

Used on a string property to indicate that the value should be hashed and salted prior to storage. Properties with this flag set are not stored in plain text, and so cannot be retrieved directly. The hashes and salts are never made available to client calls.

Values can be verified against the stored hash using Type methods. See CordraUtil Module for details.

Example:

```json
"securityQuestionAnswer": {
  "type": "string",
  "cordra": {
    "secureProperty": true
  }
}
```

disableInUi

Used on a string, number, boolean, and select properties to indicate that the UI input for this property should always be disabled. One use case for this is with a property that is calculated server side and you want to prevent users from attempting to edit it.

Example:

```json
"foo": {
  "type": "boolean",
  "cordra": {
    "disableInUi": true
  }
}
```
2.3.2 Top level properties external from the JSON Schema

2.3.2.1 indexObjects

"indexObjects": false can be added to turn off indexing of instances of this type.

2.3.2.2 indexPayloads

"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.

2.3.2.3 hashObject

See here.

2.4 Design Object

The Design object is a central location where much of the Cordra configuration is stored. For example, the configuration for the UI, authorization, and handle minting are all stored on the Design object. When logged in as a Cordra admin into the UI, the Design Object Admin menu item provides you a way to edit this object directly. The Design object could also be updated using a configuration file on startup; see repolnit.json.

Here are the properties available in the Design object for fine-tuning Cordra behavior:
<table>
<thead>
<tr>
<th>Property name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ids</td>
<td>See <em>Authentication via Keys</em>.</td>
</tr>
<tr>
<td>isReadOnly</td>
<td>While set to true, Cordra will reject object creation, update, or deletion. (Exception: the admin user, only, is allowed to edit the design object, only.)</td>
</tr>
<tr>
<td>useLegacySessionsApi</td>
<td>See <code>legacySessionsApi</code></td>
</tr>
<tr>
<td>useLegacyContentOnly-</td>
<td>See <em>Legacy JavaScript Hooks</em></td>
</tr>
<tr>
<td>JavaScriptHooks</td>
<td>If true, restores former behavior where a search with pageSize=0 returns all results. By default a search with pageSize=0 returns the number of matched objects but no object content.</td>
</tr>
<tr>
<td>useLegacySearchPageSizeZeroReturn-</td>
<td>If true, sets pageSize=0 to return all results. By default, a search with pageSize=0 returns the number of matched objects but no object content.</td>
</tr>
<tr>
<td>sAll</td>
<td>If true, restores former behavior where a search with pageSize=0 returns all results. By default a search with pageSize=0 returns the number of matched objects but no object content.</td>
</tr>
<tr>
<td>useLegacyAttributesAsInputFor-</td>
<td>By default set to false. If true, the DOIP API for HTTP Clients will copy request “attributes” into request “input” for GET requests (matching the behavior of Cordra v2.3.0 to v2.4.0).</td>
</tr>
<tr>
<td>DoipGet</td>
<td>By default set to false. If true, the DOIP API for HTTP Clients will copy request “attributes” into request “input” for GET requests (matching the behavior of Cordra v2.3.0 to v2.4.0).</td>
</tr>
<tr>
<td>enableVersionEdits</td>
<td>If true, version objects can be edited. By default, they are immutable.</td>
</tr>
<tr>
<td>includePayloadsInReplicationMessages</td>
<td>If true, payloads are included in Kafka messages produced by configuration replicationProducers. By default the payloads are omitted.</td>
</tr>
<tr>
<td>disableAuthenticationBackOff</td>
<td>By default, Cordra will slow down authentication attempts for a user after receiving an incorrect password for that user, up to a maximum of 5 seconds. Setting this to true disables the delay.</td>
</tr>
<tr>
<td>allowInsecureAuthentication</td>
<td>Allow authentication requests over HTTP. By default, only HTTPS is allowed.</td>
</tr>
<tr>
<td>enableTypeChangeViaUpdate</td>
<td>Allow standard 0.DOIP/Op.Update operation to change the type of an object.</td>
</tr>
<tr>
<td>adminPublicKey</td>
<td>JSON Web Key that can be used to log in as admin user.</td>
</tr>
<tr>
<td>uiConfig</td>
<td>See <em>UI Menu</em>.</td>
</tr>
<tr>
<td>authConfig</td>
<td>See <em>Authorization Menu</em>.</td>
</tr>
<tr>
<td>handleMintingConfig</td>
<td>See <em>Handle Integration</em>.</td>
</tr>
<tr>
<td>handleServerConfig</td>
<td>See <em>Handle Integration</em>.</td>
</tr>
<tr>
<td>doip</td>
<td>See <em>DOIP and Examples</em>.</td>
</tr>
<tr>
<td>builtInTypes.CordraDesign.javascript</td>
<td>See <em>Hooks for the Design Object and Type Objects</em>.</td>
</tr>
<tr>
<td>builtInTypes.CordraDesign.authConfig</td>
<td>See <em>Authorization</em>.</td>
</tr>
<tr>
<td>builtInTypes.Schema.javascript</td>
<td>See <em>Hooks for the Design Object and Type Objects</em>.</td>
</tr>
<tr>
<td>builtInTypes.Schema.authConfig</td>
<td>See <em>Authorization</em>.</td>
</tr>
<tr>
<td>javascript</td>
<td>See <em>Design JavaScript Menu</em>.</td>
</tr>
</tbody>
</table>
2.4.1 repoInit.json

All or part of the Design object can also be modified on startup by including a repoInit.json file in the Cordra data directory. The JSON structure under the design property in the repoInit.json file should match what is expected in the Design object. For example, to add ids that are useful for Authentication via Keys, you could include the following in the repoInit.json file:

```json
{
    "design": {
        "ids": [ "20.500.123/cordra" ]
    }
}
```

2.5 Payloads

A payload, in this context, is information in digital form that is not otherwise maintained in the JSON portion of the 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.

In Cordra, an object may have zero or more payloads. While Schemas/Types define most aspects of the information that the object manages, the presence of payloads is not specified in the JSON schema for an object, although whether or not to index payloads should be specified in that schema as discussed here.

Using the REST API, when a digital object is resolved, 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 content 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 parameters in the request.

2.6 Authentication and Access Controls

The Cordra software has a built-in access control framework. Each object may have an Access Control List (ACL) listing users and groups permitted to read and/or write the object. Additional ACLs configured in different places provide defaults and specify which users and groups may create objects of that type. Both users and groups are represented as objects, with a group being a collection of user object identifiers. Also, users and groups are not specific types in the system, but rather decorations around other types of objects that make those objects behave like users and groups for purposes of ACLs. The rest of the documentation may refer to objects as user objects and group objects, but those cases imply this nuance.

Cordra comes with pre-defined user and group objects, but it is possible to modify those or create your own. For more information, see Managing Users and Groups.
2.6.1 Authentication

Users can authenticate using passwords or public/private key pairs over the REST API or DOIP. In the REST API, users can authenticate using HTTP Basic Auth for passwords and HTTP Bearer JWT token for keys. An access token API is also available in REST. See Access Token API for more details.

The system enforces uniqueness of usernames for user objects. 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 phone numbers.

Administrators, associated with the special username admin can authenticate using passwords and/or keys. If authenticating over keys, the public key for the admin user should be managed in the Design object under a property called adminPublicKey.

**Warning:** As of Cordra v2.0.0, all authentication requests must be made over a secure HTTPS connection. To allow authentication over insecure channels, you can add “allowInsecureAuthentication”:true to the Cordra design object.

See Authenticate Hook for the lifecycle hook allowing customization of authentication.

2.6.1.1 Authentication via Passwords

User passwords are managed as part of the user objects. However, passwords are hashed and salted and stored separately from the rest of the information managed by the object. As such, the value of the password property within the JSON data is always the empty string when the object is resolved.

Password authentication (such as HTTP Basic Auth, or via Access Token API) allows specifying either a username, or the Cordra object id of the user object. (If the two possible interpretations indicate two different Cordra objects, the Cordra object id will be used.)

Cordra matches the password supplied by the user against the stored password, as part of user authentication.

2.6.1.2 Authentication via Keys

Cordra offers a more secure way to authenticate users. Users can also authenticate using public/private key pairs where the public keys are stored in either Cordra objects or Handle records.

Specifically, the public key may be stored as a JSON Web Key (JWK) as part of a user object or may be stored in an HS_PUBKEY value on a Handle record. In order to store a public key in a Cordra object that can be used for authentication, the schema for the type of object containing the public key needs the Cordra-specific schema attribute “auth”: “publicKey”. When authenticating using a public key on a Cordra object, either the username or the Cordra object id may be specified as the issuer in
the Authorization header. (If the two possible interpretations indicate two different Cordra objects, the issuer will be interpreted as a Cordra object id.) When authenticating with an HS_PUBKEY in a handle record, the handle of the record should be used as the issuer in the Authorization header. See below for how to specify the issuer syntactically.

To authenticate, users should send an Authorization header where the value of the header is Bearer followed by a serialized JSON Web Signature:

```
Authorization: Bearer eyJhbGciOiJSUzI1NiJ9.eyJ...
```

The JWS must be a self-issued JWT with at least an "iss" claim (the identity that is authenticating), and an "exp" claim at most one hour in the future (a minimum of a few minutes to avoid clock skew issues would generally suffice). If the "sub" claim is present it must be identical with the "iss" claim.

The JWT can use other safeguards to protect replay attacks. For example, the JWT can optionally include a "jti" claim by the client, so Cordra will know to prevent reuse of the same JWT for subsequent requests. (Note: Cordra instances deployed for a distributed setup do not currently support the jti claim.)

The JWT can also optionally have an "aud" claim, which contains an identifier for the Cordra instance, to prevent the reuse of the authentication claim at other Cordra instances. Cordra can be configured with identities allowed in the "aud" claim in the Design object's "ids" property. This can be added in the Cordra UI, or set on startup by including a repoInit.json file in the Cordra data directory with the following contents:

```
{
   "design": {
      "ids": [ "20.500.123/cordra" ]
   }
}
```

For more information on editing the Design object, see Design Object.

In general, the "exp", "jti", and "aud" restrictions are to prevent reuse of the authentication claims, but as long as the suggested authentication claim is sent over TLS, it is already more secure than sending a password because no secrets are exchanged in this Bearer method compared to the passwords case, but rather the possession of secrets is claimed with expiration time to bound the use of those claims indefinitely.

Example JWT for Bearer authentication:

```
{
   "iss": "20.500.123/admin",
   "sub": "20.500.123/admin",
   "jti": "5d21776da77adb89528d",
   "aud": "20.500.123/cordra",
   "exp": 1533139594
}
```

For further information about the claims used in the JWT for Cordra keypair authentication, see RFC 7519.
2.6.2 Authorization

Authorization is enabled primarily using access control lists (ACLs) as defined below. In some situations, contextual access to objects where information beyond the user or group id should be considered for providing object access. Please refer to Type Methods for leveraging lifecycle hooks to handle those special authorization situations. Furthermore, ACLs are distinguished in terms of read operations and write operations. For enabling access controls at a finer operation granularity than reads or writes, Type Methods should be leveraged.

Type level ACLs can be specified in two places: On the Type objects directly in the authConfig property or on the design authConfig object.

For ACL-based authorization, a single inheritance structure is followed: ACLs specified at the object level overrides the ACLs specified at the type level directly on the Type objects, which override type level ACLs on the design object, which override ACLs specified at the system level. Overrides are complete replacements, not merges. Priority of ACL overrides may be represented graphically as follows:

<table>
<thead>
<tr>
<th>Instance ACLs</th>
<th>Type object ACLs</th>
<th>design object type level ACLs</th>
<th>system level ACLs</th>
</tr>
</thead>
</table>

Additionally in place of specifying a JSON structure in the authConfig property directly on a Type object you can implement a JavaScript hook named `getAuthConfig` in the JavaScript for that type.

The design object (of type CordraDesign) and Type objects (of type Schema) do not have separate Type objects. In addition to the global design authConfig object, ACLs for these types can be defined under a property “builtInTypes” of the design object, specifically “builtInTypes.CordraDesign.authConfig” and “builtInTypes.Schema.authConfig”. See Design Object.

Both user object identifiers and group object identifiers play a role here. For more information on creating users and groups using the Cordra UI, see Managing Users and Groups.

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 ACL keywords below. If the ACL is left blank, then only admin can perform the operation. In the context of ACL-based authorizations, operations are categorized as only read and write.

<table>
<thead>
<tr>
<th>ACL Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>Anyone can perform the operation. They do not need to be authenticated.</td>
</tr>
<tr>
<td>authenticated</td>
<td>Any authenticated user can perform the operation. This only applies to users with user objects in Cordra, not to arbitrary handle-identified users who authenticate using public/private keypair. The handle-identified users should be explicitly given access outside of the keywords described here.</td>
</tr>
<tr>
<td>creator</td>
<td>Only the creator of the object can perform the operation.</td>
</tr>
<tr>
<td>self</td>
<td>Only a user with the same id as the object of interest can perform the operation. Typically used on defaultAclWrite setting on user objects.</td>
</tr>
</tbody>
</table>

Each object has an ACL for readers and an ACL for writers. Readers can also view the ACLs; writers can also modify the ACLs.

Authentication and Access Controls
Example:

```
{
    "readers": [
        "20.5000.215/73675debcd8a436be48e"
    ],
    "writers": [
        "20.5000.215/73675debcd8a436be48e"
    ]
}
```

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”. See Administrative APIs.

Cordra allows access to the ACLs (represented as JSON) of an object, with two properties readers and writers, each an array of strings. The type-level and default ACLs are configured by specifying a JSON structure as well.

Example:

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

**NOTE:** The JSON representation of ACL has changed in Cordra v2. In v1, they were called read and write. In v2, the properties are called readers and writers respectively.
2.6.2.1 Authorization for Type Methods

Type methods are Cordra's way of enabling custom operations to be added to the system. And these operations or methods can be enabled in the context of a given type; hence the name Type methods. See Type Methods for more details.

In addition to restricting who can read or write to an object, ACLs can also be defined who can execute these methods. ACLs for calling Type methods can be defined on the "schemaAcls" or "defaultAcls" of the global authorization configuration. The property "aclMethods" determines the authorization. If "aclMethods" is missing on the "schemaAcls" entry for the type in question, or on the "defaultAcls" if there is no such entry, then Cordra assumes that all authorized writers of the instance object (for an instance method) or the schema/type object (for a static method) are authorized to call the method.

The property "aclMethods" is an object with properties "instance" and "static", each a map from method name to ACL, as well as a property "default", an object with properties "instance" and "static", each an ACL. The default method ACLs apply to methods which are not named explicitly under "instance" or "static".

Method ACLs can use the additional keywords "readers" and "writers", which authorize respectively all the authorized readers or writers of the instance object (for an instance method) or schema object (for a static method). In the absence of any "aclMethods" entry, all methods are considered to have ACL [ "writers" ]. If "aclMethods" is defined but a method is missing and no default is defined, it is considered to have ACL [ ] (admin access only).

Example:

```json
{
    "schemaAcls": {
        "User": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [ "self" ],
            "aclCreate": []
        },
        "Document": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [ "creator" ],
            "aclCreate": [ "public" ],
            "aclMethods": {
                "static": {
                    "exampleStaticMethod": [ "public" ]
                },
                "instance": {
                    "exampleInstanceMethod": [ "authenticated" ]
                },
                "default": {
                    "instance": [ "writers" ]
                }
            }
        }
    }
}
```

(continues on next page)
In this example, static method "exampleStaticMethod" on type Document can be called by anyone, even anonymously; instance method "exampleInstanceMethod" on objects of type Document can be called by any authenticated user. Other instance methods on objects of type Document can only be called by authorized writers of the object. Since the "static" property on "default" is missing, other static methods can only be called by admin. On types other than Document, all methods can be called by authorized writers.

An ACL for a specific instance of an object can define the callers of a method. Specifying such a methods ACL on an object overrides type-level method ACLs for the methods specified. Other methods will still use type-level method ACLs.

Example:

```json
{
    "readers": [
        "public"
    ],
    "writers": [
        "20.5000.215/73675debc1a436be48e"
    ],
    "methods": {
        "exampleInstanceMethod": [
            "20.5000.215/73675debc1a436be48e"
        ]
    }
}
```

If there is no aclMethods defined at the type level for an object writers are permitted to call any operation on an object instance.

If there are writers defined at the instance level and there are callers for a method defined at the instance level then the method callers are permitted to call the method even if they are not in the writers list.
2.6.2.2 Restricting Access to Payloads

The property `payloadReaders` can also be set on the instance ACLs. If `payloadReaders` is missing then any user who is permitted to read or write to the object has implied permission to also read the payload. However if `payloadReaders` is not missing then in order to read a payload a user must be in the `payloadReaders` list AND be permitted to read or write to the object.

Example:

```json
{
    "readers": [
        "test/user2"
    ],
    "writers": [
        "test/user1",
    ],
    "payloadReaders": [
        "test/user1"
    ]
}
```

In the above example `test/user1` and `test/user2` can read the object, `test/user1` can write to the object and only `test/user1` can read the bytes of the payload attached to the object.

It is also possible restrict access to payloads at the type level by setting the property `defaultAclPayloadRead` as a sibling to the other type level ACLs.

2.7 Lifecycle Hooks

Cordra validates incoming information against schemas as defined in the Type objects. Additional rules that validate and/or enrich the information in the object can be configured to be applied by Cordra at various stages of the object lifecycle. Such rules are expected to be specified in JavaScript and to be bound to the following lifecycle points:

- before an object is schema validated (with an opportunity, during create, to set the id or properties which help determine the id);
- during id generation;
- before schema validation but after id generation;
- before an object is stored, allowing for additional validation or side-effects after all other validation is done;
- before an object is indexed, allowing the object that is indexed to differ from the one that is stored;
- during handle record creation;
- after an object (or payload) has been retrieved from storage, but before it is returned;
• before an object is deleted, to forbid deletion under some circumstances;
• after an object is deleted;
• after an object is created or updated; and
• before executing a user-supplied query.

Lifecycle hooks, in our parlance, are the points of entry for your own JavaScript-based rules that Cordra processes. In addition to the lifecycle hooks that are discussed in detail below, Cordra enables clients to invoke other rules in an ad hoc fashion using Type Methods.

Currently, various lifecycle hooks are enabled in Cordra for different actions: create, update, retrieve, and delete.

The following diagrams illustrate hooks that are enabled during various stages of the object lifecycle.
Cordra Update Lifecycle

Authenticate
Authorize
Acquire Id Lock
Get from storage
Schema Validate
Update hashes
Update storage
Index
Send replication
Update cache
Update handle
Release Id Lock
Send response

beforeSchemaValidation
beforeSchemaValidationWithId
beforeStorage
objectForIndexing
createHandleValues
afterCreateOrUpdate
onObjectResolution

JavaScript Hooks
Cordra operation
Several internal operations are not included for brevity.
Cordra Retrieve Lifecycle

1. Authenticate
2. Authorize
3. Get from storage
4. onObjectResolution
5. onPayloadResolution
6. Send response

Cordra Delete Lifecycle

1. Authenticate
2. Authorize
3. Acquire Id Lock
4. Get from storage
5. beforeDelete
6. Delete in storage
7. Update cache
8. Delete handle
9. Delete in Index
10. Send replication
11. afterDelete
12. Release Id Lock
13. Send response

JavaScript Hooks

Cordra operation

Several internal operations are not included for brevity.
An update which changes the type of an object will run the beforeDelete and afterDelete hooks of the original type, only with a context where "isUpdate" is set to true, as well as the beforeSchemaValidation, beforeStorage, and afterCreateOrUpdate hooks of the new type, with "isNew" set to true (like a creation) but also with "isUpdate" set to true.

### 2.7.1 Using Hooks in JavaScript

Hooks can either be implemented in the "javascript" property of the design object, which makes a "service-level hook", or hooks can be implemented on a per type basis as part of the JavaScript associated with each Type object. When a lifecycle hook is implemented as a service-level hook that hook will apply to all objects in the system. A hook implemented in JavaScript associated with a Type object will only apply to objects of that type. Hooks implemented for a specific type override the same hook implemented as a service-level hook. For example if you have `onObjectResolution` in design JavaScript and you have `onObjectResolution` implemented in JavaScript for type "Foo", the service-level hook will be used for all objects except those of type "Foo" which will use the type specific implementation.

**Warning:** In versions of Cordra before 2.5.0, implementing service-level lifecycle hooks in design JavaScript did not mean that hook would be applied to all object types. In 2.5.0 hooks implemented in design JavaScript apply to all objects and can be overridden at the type level.

### 2.7.1.1 Hooks in a Type Object

Most lifecycle hooks are available for use as part of the JavaScript associated with each Type object, in the "javascript" property of the schema object content. To have have the same code that runs for multiple type you can either implement the hook as a service-level hook or you will need to have JavaScript for each of the types. See *Using External Modules* for methods to share code among multiple types.

The examples in this documentation are written as CommonJS modules, but Cordra also supports ECMAScript modules; see *ECMAScript Modules*.

Here is the shell of the hooks that are available in each Type, which will be explained below.

```javascript
1 const cordra = require('cordra');
2 const cordraUtil = require('cordra-util'); // older name 'cordraUtil' also works
3 const { CordraClient, Blob } = require('cordra-client');
4 const schema = require('/cordra/schemas/Type.schema.json');
5 const js = require('/cordra/schemas/Type');
6
7 exports.beforeSchemaValidation = beforeSchemaValidation;
8 exports.beforeSchemaValidationWithId = beforeSchemaValidationWithId;
9 exports.beforeStorage = beforeStorage;
10 exports.objectForIndexing = objectForIndexing;
11 exports.onObjectResolution = onObjectResolution;
```

(continues on next page)
exports.onPayloadResolution = onPayloadResolution;
exports.beforeDelete = beforeDelete;
exports.afterDelete = afterDelete;
exports.afterCreateOrUpdate = afterCreateOrUpdate;
exports.getAuthConfig = getAuthConfig;

function beforeSchemaValidation(object, context) {
  /* Insert code here */
  return object;
}

function beforeSchemaValidationWithId(object, context) {
  /* Insert code here */
  return object;
}

function beforeStorage(object, context) {
  /* Insert code here */
}

function objectForIndexing(object, context) {
  /* Insert code here */
  return object;
}

function onObjectResolution(object, context) {
  /* Insert code here */
  return object;
}

function onPayloadResolution(object, context) {
  /* Insert code here; use context.directIo to write payload bytes */
}

function beforeDelete(object, context) {
  /* Insert code here */
}

function afterDelete(object, context) {
  /* Insert code here */
}

function afterCreateOrUpdate(object, context) {
  /* Insert code here */
}
function getAuthConfig(context) {
    /* Insert code here */
}

Cordra provides three convenience JavaScript modules that can be imported for use within your JavaScript rules. These modules allow you to search and retrieve objects, verify hashes and secrets, and perform arbitrary operations against the host Cordra as arbitrary authenticated users. Additional modules allow you to retrieve schemas and associated JavaScript hooks, as discussed here. You can optionally include these modules in your JavaScript, as shown on Lines 1-5.

You can also save external JavaScript libraries in Cordra for applying complex logic as discussed here.

Lines 7-15 export references to the hooks that Cordra enables on a Type object: beforeSchemaValidation, beforeSchemaValidationWithId, beforeStorage, objectForIndexing, onObjectResolution, onPayloadResolution, beforeDelete, afterDelete and afterUpdateOrCreate,. When handling objects, Cordra will look for methods with these names and run them if found. The methods must be exported in order for Cordra to see them. None of the methods is mandatory. You only need to implement the ones you want.

Resolution of payloads will activate both onObjectResolution and onPayloadResolution. If onPayloadResolution accesses the output via context.directIo (see Direct I/O) the hook will fully control the output bytes, and the stored payload will not be directly returned to the client.

The rest of the example shell shows the boilerplate for the methods. All, except for getAuthConfig, take both an object and a context. object is the JSON representation of the Cordra object. It may be modified and returned by beforeSchemaValidation, beforeSchemaValidationWithId, objectForIndexing, and onObjectResolution.

object contains id, type, content, acl, metadata, and payloads (which has payload metadata, not the full payload data). content is the user defined JSON of the object.

object has the following format:

```json
{
    "id": "test/abc",
    "type": "Document",
    "content": { },
    "acl": { 
        "readers": [
            "test/user1",
            "test/user2"
        ],
        "writers": [
            "test/user1"
        ]
    },
    "metadata": {
        "createdOn": 1532638382843,
```
context is an object with several useful properties.
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNew</td>
<td>Flag which is true for creations and false for modifications. Applies to beforeSchemaValidation.</td>
</tr>
<tr>
<td>objec-tld</td>
<td>The id of the object.</td>
</tr>
<tr>
<td>userId</td>
<td>The id of the user performing the operation.</td>
</tr>
<tr>
<td>groups</td>
<td>A list of the ids of groups to which the user belongs.</td>
</tr>
<tr>
<td>grantAuthenti-catedAccess</td>
<td>Flag which is true when the user performing the operation is considered “authenticated” for the purpose of the “authenticated” ACL keyword (generally true when the user has a user object).</td>
</tr>
<tr>
<td>auth-Context</td>
<td>Arbitrary information added by the “authenticate” hook (see Authenticate Hook).</td>
</tr>
<tr>
<td>effec-tiveAcl</td>
<td>The computed ACLs for the object, either from the object itself or inherited from configuration. This is an object with &quot;readers&quot; and &quot;writers&quot; properties.</td>
</tr>
<tr>
<td>aclCreate</td>
<td>The creation ACL for the type being created, in beforeSchemaValidation for a creation.</td>
</tr>
<tr>
<td>new-Payloads</td>
<td>A list of payload metadata for payloads being updated, in beforeSchemaValidation for an update operation.</td>
</tr>
<tr>
<td>payload-ToDelete</td>
<td>A list of payload names of payloads being deleted, in beforeSchemaValidation for an update operation.</td>
</tr>
<tr>
<td>request-Context</td>
<td>A user-supplied requestContext query parameter.</td>
</tr>
<tr>
<td>payload</td>
<td>The payload name for onPayloadResolution.</td>
</tr>
<tr>
<td>start, end</td>
<td>User-supplied start and end points for a partial payload resolution for onPayloadResolution.</td>
</tr>
<tr>
<td>params</td>
<td>The input supplied to a Type Methods call. Can be used for more control over input/output with Type Methods or onPayloadResolution; see Direct I/O.</td>
</tr>
<tr>
<td>direc-tio</td>
<td>Request attributes supplied with a DOIP request or a method call (see Request Attributes).</td>
</tr>
<tr>
<td>attributes</td>
<td>Flag set to true in onObjectResolution if the call is being made to produce search results.</td>
</tr>
<tr>
<td>is-Search</td>
<td>Flag set to true in onObjectResolution if the call is being made with the result of creating a Cordra object; set to true in beforeSchemaValidation, beforeStorage, afterCreateOrUpdate for creations.</td>
</tr>
<tr>
<td>isCre-ate</td>
<td>Flag set to true in onObjectResolution if the call is being made with the result of updating a Cordra object; set to true in beforeSchemaValidation, beforeStorage, afterCreateOrUpdate for updates; set to true when beforeDelete and afterDelete are called when an update changes an object’s type (this is treated like a deletion followed by a creation).</td>
</tr>
<tr>
<td>isUp-date</td>
<td>Flag set to true in onObjectResolution if the call is being made with the result of updating a Cordra object; set to true in beforeSchemaValidation, beforeStorage, afterCreateOrUpdate for updates; set to true when beforeDelete and afterDelete are called when an update changes an object’s type (this is treated like a deletion followed by a creation).</td>
</tr>
<tr>
<td>DryRun</td>
<td>Set on a create or update according to the “dryRun” parameter. Could be used to prevent external side effects.</td>
</tr>
<tr>
<td>method</td>
<td>Set to the method name in beforeStorage or afterCreateOrUpdate when activated after a failed create or update.</td>
</tr>
</tbody>
</table>
2.7.1.2 Before Storage Hook

This beforeStorage hook is run immediately before the object is inserted into storage. It has gone through all other processing and contains the generated id if an id was not included in the request. There is no return value for this function. As such you cannot edit the object here; however, you can perform additional validation and throw an exception if you want to reject the object.

2.7.1.3 Generate Object Id Hook

This hook is for generating object ids when objects are created.

The shell for this hook is as follows:

```javascript
exports.generateId = generateId;
exports.isGenerateIdLoopable = true;

function generateId(object, context) {
    var id;
    /* Insert code here */
    return id;
}
```

The flag isGenerateIdLoopable when set to true tells Cordra that if an object with the same id already exists this method can be called repeatedly until a unique id is found. If the implementation of generateId was deterministic, which is to say it would always return the same id for a given input object, the isGenerateIdLoopable should NOT be set to true.

If null or undefined or the empty string is returned, Cordra will use its default identifier-minting logic.

**Warning:** In versions of Cordra before 2.4.0, generateId was guaranteed to run after schema validation. Cordra 2.4.0 introduced beforeSchemaValidationWithId, which requires generateId to run before schema validation. In order to preserve backward compatibility, generateId will currently still run after schema validation when beforeSchemaValidationWithId is unused; however, this flow should be considered deprecated and should not be relied on when writing new hook code. New generateId code should be robust to input which is not schema-valid, although throwing or returning null or undefined would suffice.

2.7.1.4 Hooks for the Design Object and Type Objects

The design object (of type CordraDesign) and Type objects (of type Schema) do not have separate Type objects. These built-in types can still have lifecycle hooks, however. Their JavaScript modules can be defined under a property “builtinTypes” of the design object, specifically “builtinTypes.CordraDesign.javascript” and “builtinTypes.Schema.javascript”. See *Design Object*. 

Lifecycle Hooks
2.7.1.5 Get Auth Config Hook

This `getAuthConfig` hook is unlike other hooks defined on a Type object in that it is not called with an instance of an object of that type.

The shell for this hook is as follows:

```javascript
exports.getAuthConfig = getAuthConfig;

function getAuthConfig(context) {
    return {
        "defaultAclRead": [
            "test/group1"
        ],
        "defaultAclWrite": [
            "test/group1"
        ],
        "aclCreate": [
            "test/group1"
        ]
    };
}
```

2.7.1.6 Authenticate Hook

This hook is also looked for in the “javascript” property of the design object. It will be executed for every user request that requires authentication.

See External Authentication Provider for a substantial example of how this hook could be used.

The shell for this hook is as follows:

```javascript
exports.authenticate = authenticate;

function authenticate(authInfo, context) {
    /* Insert code here */

    return { /* ... */ };
}
```

The `authInfo` passed into this function has the following structure:

```javascript
{
    "username": "",
    "password": "",
    "token": "",
    "authHeader": "",
    "authTokenInput": {}
}
```
"doipAuthentication": {},
"doipClientId": "",
"asUserId": ""
}

The hook can inspect “username” and “password”, or “token”, for typical authentication scenarios; but the hook also has access to the raw authentication data in various forms should it be needed.

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>String. The username from a Basic HTTP Authorization header, or from username/password input to /auth/token or DOIP authentication.</td>
</tr>
<tr>
<td>password</td>
<td>String. The password from a Basic HTTP Authorization header, or from username/password input to /auth/token or DOIP authentication.</td>
</tr>
<tr>
<td>token</td>
<td>String. The Bearer token from a Bearer HTTP Authorization header, or a JWT assertion made as /auth/token input, or a token from DOIP authentication.</td>
</tr>
<tr>
<td>authHeader</td>
<td>String. The value of the request's HTTP Authorization header.</td>
</tr>
<tr>
<td>doipAuthentication</td>
<td>Object. If the request came in over the DOIP interface this property will contain the value of the passed-in DOIP authentication.</td>
</tr>
<tr>
<td>doipClientId</td>
<td>String. If the request came in over the DOIP interface this property will contain the clientId, if available.</td>
</tr>
<tr>
<td>asUserId</td>
<td>String. The asUserId property may be optionally set to indicate that the call should should be performed with the permissions that user has been granted; see As-User. Many authenticate hook use cases can ignore this. Default Cordra authentication logic only allows admin to use As-User; the authenticate hook can handle the As-User request by returning a userId which has the value of the provided asUserId, and can allow users other than admin as desired. Only set the output userId to the input asUserId when you really want to allow the authenticating user the full privileges of the asUserId. If the hook return userId does not match the input asUserId, the default As-User logic will occur; to prevent even admin use of As-User, throw an exception. Note that if this property is set, and you are returning custom groupIds which you wish to be available using As-User, you should set userId to the asUserId, and return the groupIds for the asUserId.</td>
</tr>
</tbody>
</table>

Returning null or undefined causes the default cordra authentication code to be executed in that

Lifecycle Hooks 57
To reject an authentication attempt throw an exception.

To accept an authentication return a result object. The result object should contain at least
"active": true and the userId. It can optionally contain username and groupIds, where groupIds
are the groups the user is a member of.

The active property on the result object should be set to true. If it is set to false it will cause an
exception to be thrown.

**Warning:** Note that the userId “admin” will be given the full privileges of the Cordra admin user.
Thus take care that the behavior of any external authentication provider used by the authenticate
hook does not give users full control over their own userIds.

An example successful authentication result object is shown below:

```json
{
   "active": true,
   "userId": "test/123",
   "username": "fred",
   "groupIds": [ "test/group1" ],
   "grantAuthenticatedAccess": true,
   "bypassCordraGroupObjects": true
}
```
<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>Boolean. Set to true if the authentication is successful.</td>
</tr>
<tr>
<td>userId</td>
<td>String. The userId of the authenticated user.</td>
</tr>
<tr>
<td>username</td>
<td>String (Optional). The username of the authenticated user.</td>
</tr>
<tr>
<td>groupIds</td>
<td>List of String (Optional). Custom groupIds the authenticated user is a member of.</td>
</tr>
<tr>
<td>bypassCordraGroupObjects</td>
<td>Boolean (Optional) Used in combination with groupIds. If set to true only the returned groupIds will be used. If missing or set to false any Cordra group objects that contain this users id will be combined with the custom groupIds.</td>
</tr>
<tr>
<td>grantAuthenticatedAccess</td>
<td>Boolean (Optional). If set to true this user is considered “authenticated” by ACLs that use that keyword. If missing Cordra ACLs will only consider this user “authenticated” if their exists a Cordra object that corresponds with their userId. If a hook allows authentication by a wide open set of users, such as any users with an account with some existing global service, then it would be appropriate to set this false. If your ACLs do not use the “authenticated” keyword, this has no effect.</td>
</tr>
<tr>
<td>exp</td>
<td>Number (Optional). Expiration in Epoch Unix Timestamp, the number of seconds since JAN 01 1970. (UTC) that the session should last for before expiration. If not supplied by default the session will not expire.</td>
</tr>
<tr>
<td>authContext</td>
<td>Arbitrary JSON (Optional). The value of authContext is provided to all lifecycle hooks (as part of the context) for operations making use of this authentication.</td>
</tr>
</tbody>
</table>

There is one additional special response which the authenticate hook can return:

```json
{
    "anonymous": true
}
```

This response causes Cordra to behave as if there was no authentication attempt.

**Warning:** It is possible to lock out all users, including admin, by throwing an exception from the authenticate hook. Doing so would then prevent admin from signing in to edit the problem JavaScript. To recover from such a situation place the following example repoInit.json file in the data directory. Upon restart, this will delete all the JavaScript on the design object allowing admin to sign in to fix it.

```json
{
    "design": {
        "javascript": ""
    }
}
```
2.7.1.7 Customize Query and Params Hook

This hook is also looked for in the “javascript” property of the design object. It will be executed for every user-supplied query to the Search API. It can be used for example to restrict the query to exclude certain objects based on the calling user. It can also be used to modify the query params supplied by the user for example to restrict the number of results per page, or to throw an exception when a user uses search parameters which are to be disallowed to that user. Query restriction can be accomplished either by altering the query or adding filterQueries (see Search for objects).

The shell for this hook is as follows:

```javascript
exports.customizeQueryAndParams = customizeQueryAndParams;

function customizeQueryAndParams(queryAndParams, context) {
    /* Insert code here */
    queryAndParams.query = newQuery;
    queryAndParams.pageSize = 5;
    return queryAndParams;
}
```

2.7.1.8 Customize Query Hook

This hook is similar to the above-mentioned customizeQueryAndParams but only allows customization of the query string. If hooks customizeQueryAndParams and customizeQuery are both implemented, both will be executed with customizeQuery being run after.

The shell for this hook is as follows:

```javascript
exports.customizeQuery = customizeQuery;

function customizeQuery(query, context) {
    /* Insert code here */
    return newQuery;
}
```

2.7.1.9 Create Handle Values Hook

This hook is for specifying the handle record that is to be returned when handles are resolved using handle client tools. This hook can be defined with other service-level hooks in Design JavaScript; it can alternatively be defined on the separate design object property design.handleMintingConfig.javascript, which can be edited by selecting Handle Records from the Admin menu on the UI.

The shell for this hook is as follows:

```javascript
exports.createHandleValues = createHandleValues;

function createHandleValues(object, context) {
    (continues on next page)
```
If creating handle values with JavaScript it is important to consider that all Cordra objects, even if not publicly visible, will have a handle record created. If you are storing data directly in the handle record you may wish to check if the Cordra object is publicly accessible. You can do this by inspecting the 'context' argument. For example:

```javascript
function isPublic(context) {
    const effectiveAcl = context.effectiveAcl;
    if (effectiveAcl.writers && effectiveAcl.writers.indexOf("public") !== -1) {
        return true;
    } else if (effectiveAcl.readers && effectiveAcl.readers.indexOf("public") !== -1) {
        return true;
    } else {
        return false;
    }
}
```

### 2.7.2 Asynchronous Lifecycle Hooks

The return value of any hook (or type method) can be a JavaScript promise. When a hook returns a promise, Cordra will wait until the promise is fulfilled or rejected. If the promise is fulfilled, Cordra will use the value to which the promise resolved. If the promise is rejected, Cordra will treat the rejection reason like it would a thrown exception. This facility makes it possible to use asynchronous libraries to write Cordra hooks and type methods.

### 2.7.3 Throwing Errors in Schema JavaScript

Errors thrown as strings will end up in the server response, with the thrown string as the error message.

```javascript
throw "An error has occurred";
```

If the user requests are issued via the REST API, for beforeSchemaValidation and Type methods calls, this will be returned to the user as a 400 Bad Request. For onObjectResolution and beforeDelete, this will be returned as 403 Forbidden. For search results where onObjectResolution throws an exception, the corresponding object will be omitted from the search results (this can affect search results count).

If the user requests are issued via the DOIP interface, a “bad request” or “forbidden” error will be returned.
For more control over the server response, you can also throw a custom CordraError, available via the Cordra.js Module. For example:

```javascript
const cordra = require("cordra");

const responseCode = 418; // defaults to 400 or (for resolution) 403 if undefined
const response = {
  message: "Beverage Not Supported",
  requestedBeverage: "coffee",
  supportedBeverages: ["tea", "water"]
};
throw new cordra.CordraError(response, responseCode);
```

This will be translated into a server response with the given error message and response status code. If present, the response object will be added to the body of the server response. This can be used to send extra information about the error back to the caller.

As a convenience, if the first argument to new cordra.CordraError is a string, the response will be {"message": "that string"}. The first object can also be an instance of JavaScript Error or a Java Throwable. If there is no responseCode given, a CordraError or CordraException will be propagated using the given object's response code, and any other Error or Throwable will be propagated with 500 Internal Server Error.

The constructor for cordra.CordraError can take an options argument as its last parameter, which will be given to the JavaScript Error constructor. This can be used to set up a cause property in the ordinary way. If the first argument to the constructor is an Error or Throwable, that will be the cause if no options argument is present.

Thrown errors other than strings and CordraError will be seen by the user as 500 Internal Server Error.

### 2.7.4 Cordra Modules

#### 2.7.4.1 Cordra.js Module

The builtin Cordra.js module has helpful functions, listed below, that may be useful when writing JavaScript code in Type methods.

See also the Cordra Client Module which has a more general API for performing operations against the host Cordra.

**Note:** Lifecycle hooks are triggered when calls are made using the external APIs. Calls made to Cordra using the helpful functions in the cordra.js module do not trigger any lifecycle hooks.
Search

Use the search function to find objects in the Cordra index:

```java
cordra.search(query, params)
```

This will return an array (in JSON sense) of Cordra objects matching the query. `params` is an optional object with optional fields `pageNum`, `pageSize`, `sortFields`, `facets`, `filterQueries`, `includeScore`, and `includeVersions`.

The default behavior is to get all results for a query, which corresponds to a `pageNum` of 0 and `pageSize` of -1. Caution should be used when requesting all results when the query might match a very large number of objects.

The legacy form `cordra.search(query, pageNum, pageSize, sortFields)` still functions as well.

Note: Former versions of Cordra would return all results with `pageSize=0`. To restore this former behavior, you can add "useLegacySearchPageSizeZeroReturnsAll":true to the Cordra Design object. By default a search with `pageSize=0` returns the number of matched objects but no object content.

Get

Use get to get an object from Cordra by the object ID:

```java
cordra.get(objectId);
```

If an object with the given ID is found, it will be returned. Otherwise, `null` will be returned.

Payload Retrieval

Use any of the following to retrieve a payload from Cordra using the object ID and the payload name:

```java
cordra.getPayloadAsJavaInputStream(objectId, payloadName);
cordra.getPayloadAsJavaReader(objectId, payloadName);
cordra.getPayloadAsUint8Array(objectId, payloadName);
cordra.getPayloadAsString(objectId, payloadName);
cordra.getPartialPayloadAsJavaInputStream(objectId, payloadName, start, end);
cordra.getPartialPayloadAsJavaReader(objectId, payloadName, start, end);
cordra.getPartialPayloadAsUint8Array(objectId, payloadName, start, end);
cordra.getPartialPayloadAsString(objectId, payloadName, start, end);
```
CordraError

See *Throwing Errors in Schema JavaScript*.

2.7.4.2 CordraUtil Module

**Escape for Query**

Will modify a string for literal inclusion in a phrase query for calling `cordra.search`:

```javascript
const query = /property:"" + cordraUtil.escapeForQuery(s) + '"';
```

**Verify Secret**

Used to verify a given string against the hash stored for that property:

```javascript
cordraUtil.verifySecret(obj, jsonPointer, secretToVerify);
```

Return true or false, depending on the results of the verification.

**Verify Hashes**

Verifies the hashes on a cordra object property:

```javascript
cordraUtil.verifyHashes(obj);
```

Returns a verification report object indicating which of the object hashes verify.

**Hash Json**

Hashes a JSON object, JSON array or primitive:

```javascript
cordraUtil.hashJson(jsonElement);
cordraUtil.hashJson(jsonElement, algorithm);
```

Returns a base16 encoded string of the SHA-256 hash (or other specified algorithm) of the input. The input JSON is first canonicalized before being hashed.
Sign With Key

Signs a payload (a string) with a given private key in JWK format:

```javascript
const jws = cordraUtil.signWithKey(payload, jwk);
```

Returns a Json Web Signature in compact serialization.

Sign With Cordra Key

Signs a payload (a string) with the private key of the Cordra instance:

```javascript
const jws = cordraUtil.signWithCordraKey(payload);
```

Returns a Json Web Signature in compact serialization.

The private key used is the same key used for administering an external handle server, and can be set by including a file “privatekey” in the Cordra data directory. See Handle Server and also Cordra Configuration for the distributed version.

Retrieve Cordra Public Key

Returns the Cordra public key in JWK format:

```javascript
const jwk = cordraUtil.getCordraPublicKey();
```

Verify With Cordra Key

Verifies a JWS with the private key of the Cordra instance:

```javascript
const isValid = cordraUtil.verifyWithCordraKey(jws);
```

Verify With Key

Verifies a JWS with the supplied private key in JWK format:

```javascript
const isValid = cordraUtil.verifyWithKey(jws, jwk);`
Extract JWT Payload

Extracts the payload of a JWT, returning the parsed JSON as a JavaScript object:

```javascript
const claimsObject = cordraUtil.extractJwtPayload(jws);
```

Get Groups for User

Given a user id, returns a list of group ids, which are the Cordra group objects that reference the user id. Does not interact with custom groups provided by the response of the “authenticate” hook.

```javascript
const groupIds = cordraUtil.getGroupsForUser(user);
```

Validate With Schema

Given some variable `input` and a JSON schema, this function will validate the input against the JSON schema and return a report:

```javascript
const input = {
    name: "foo",
    description: "bar"
};

const jsonSchema = {
    type: "object",
    required: [
        "name",
        "description"
    ],
    properties: {
        name: {
            type: "string"
        },
        description: {
            type: "string"
        }
    }
};

const report = cordraUtil.validateWithSchema(input, jsonSchema);
```

The resulting report will contain a boolean `success` and possibly a list of errors, where each error will have a `message` and possibly a `pointer` indicating where in the JSON the error occurs.

The JsonSchema may use `$ref` to reference other schemas in Cordra. For example if you have a Cordra type named `Foo` you could do the following:
const input = {
    name: "foo",
    description: "bar"
};

const jsonSchema = {
    "$ref": "Foo"
};

const report = cordraUtil.validateWithSchema(input, jsonSchema);

### 2.7.4.3 Cordra Client Module

Using `require('cordra-client')` or `import ... from 'cordra-client'` allows access to APIs for interacting with the host Cordra, as arbitrary authenticated users. The module APIs are largely compatible with the [*@cnri/cordra-client* JavaScript library.](https://github.com/cordra/cordra-client)

### Import

To import, use either:

```javascript
const { CordraClient, Blob } = require('cordra-client');
```

or:

```javascript
import CordraClient, { Blob } from 'cordra-client';
```

Note that the `Blob` import is used to interact with payloads and binary schema methods, and is otherwise not necessary.

### Construction, Options, Authentication

To create a new `CordraClient` which will interact as the admin user:

```javascript
const client = new CordraClient();
```

The `CordraClient` constructor takes a single argument which is the default Options for calls made using the client. Individual calls can specify the Options instead of using the default. The Options can specify an attempt to authenticate (such as `{ "username": "...", "password": "..." }`), but a helper static method allows creating an Options which is automatically authenticated as a specified userId:

```javascript
const client = new CordraClient(CordraClient.optionsForUserId('123/abc'));
```
API Calls

Once you have an instance of CordraClient, call it exactly as described in the API for the cordra-client JavaScript library. In particular, calls are generally asynchronous and use of async/await is recommended.

Binary Blobs

For interacting with payloads and binary schema methods, the CordraClient APIs use the Blob type. This Blob supports methods blob.arrayBuffer() and blob.text() which return a Promise to an ArrayBuffer and a Promise to a string, respectively; it additionally supports blob.javaInputStream() which returns (synchronously) a Java InputStream instance. Additionally, a Blob can be constructed from a Java InputStream using:

```
new Blob([javaInputStream]);
```

Restrictions on Locking (Write) Interactions

Note that some JavaScript is run in a context where the Cordra object is locked against changes from multiple simultaneous calls. In the locked context it is not allowed to create, update, delete, or call instance methods on other Cordra objects – those interactions require another lock, which could lead to deadlock. To prevent this, CordraClient will throw an error when such a call is made inside the lock. Get, search, and static schema methods are always safe to call. From inside of static methods it is safe to perform locking (write) interactions; from the hooks afterCreateOrUpdate and afterDelete it is safe to perform locking (write) interactions. It is generally either not possible or not desirable to perform locking (write) interactions from inside instance methods or other hooks; see the figures at Lifecycle Hooks for details on which hooks are inside the lock.

2.7.4.4 Cordra Schemas and JavaScript

Schemas associated with type objects are available to the JavaScript via require('/cordra/schemas/Type.schema.json'), and JavaScript added to those type objects via require('/cordra/schemas/Type'). Here Type should be replaced by the name of the particular type to be accessed.

2.7.5 Using External Modules

Note: one simple way to share JavaScript code is to have one schema’s JavaScript access another schema’s JavaScript, as outlined in Cordra Schemas and JavaScript. Since the JavaScript can export properties besides the hooks and methods, any code can be shared this way.

Additionally, external JavaScript modules can be managed with a Cordra object as a payload configured to be a “referrable” source of JavaScript modules. Typically, this can be done on a single object of a type called JavaScriptDirectory. Here are the steps needed to create and populate the JavaScriptDirectory object.
1. Create a new schema in Cordra called "JavaScriptDirectory" and using the "javascript-directory" template.

2. Create a new JavaScriptDirectory object. Set the directory to /node_modules. This will allow you to import modules by filename, instead of directory path and filename.

3. Add your JavaScript module files as payloads to the JavaScriptDirectory object. The payload name should match the filename and will be used when importing a module. For example, a payload named util.js could be importing using require('util');

The use of external JavaScript modules affects reindexing. It is currently necessary to ensure that objects of type "Schema" and any sources of JavaScript (like type "JavaScriptDirectory") are indexed first. See Reindexing for information.

2.7.6 JavaScript Version

By default Cordra uses the GraalVM JavaScript engine. This allows JavaScript features up to ECMAScript 2022.

In case your JavaScript requires Nashorn, you can configure Cordra to continue using Nashorn by adding to config.json:

```json
"javascript": {
   "engine": "nashorn"
}
```

Nashorn supports ECMAScript 5.1 but Cordra JavaScript using Nashorn does come prepopulated with a wide range of polyfills allowing features up to ECMAScript 2017. It is thus in many cases straightforward to write ECMAScript 2017 code and transpile it (using for example Babel) to ES5 for use in Cordra.

2.7.7 ECMAScript Modules

The examples in this documentation are written as CommonJS (CJS) modules, using require to import functionality and setting properties of exports to export functionality. Cordra also supports using ECMAScript (ESM) modules, which use import to import functionality and export to export functionality.

By default, Cordra will access JavaScript using CJS. If you have written an ESM module, you must set the "javascriptIsModule" property (a sibling of the "javascript" property) to true. This property exists on Type objects, and on the design object everywhere JavaScript can be included.

For shared code (Using External Modules) the type of a module must be correctly set on the caller side, that is, the caller must either require or import the module, as appropriate.

The built-in modules "cordra", "cordra-client", and "cordra-util" (also available for backward compatibility as "cordraUtil") can be either required or imported.
2.7.8 Legacy JavaScript Hooks

In early versions of the Cordra 2.0 Beta software, the JavaScript hooks `beforeSchemaValidation`, `onObjectResolution`, and `beforeDelete` took the JSON content of the Cordra object, instead of the full Cordra object (including id, type, content, acl, metadata, and payloads properties). Additionally, the JavaScript `cordra.get` function returned only the content instead of the full Cordra object.

If a Cordra instance with JavaScript written for those earlier versions needs to be upgraded, and it is not yet possible to adapt the JavaScript to the current API, then the following flag must be added to the Design object:

```
"useLegacyContentOnlyJavaScriptHooks": true
```

For more information on editing the Design object, see [Design Object](#).

Cordra users upgrading from early versions of the Cordra 2.0 beta, who did not use schema JavaScript (apart from the default User schema JavaScript, which will be automatically upgraded if it has not been edited), do not in general need to take any action.

2.7.9 Examples of Hooks

2.7.9.1 Example: User Schema JavaScript

The default Cordra User schema comes with JavaScript that performs basic password validation.

```javascript
const cordra = require("cordra");

exports.beforeSchemaValidation = beforeSchemaValidation;

function beforeSchemaValidation(object, context) {
    if (!object.content.id) object.content.id = ";
    if (!object.content.password) object.content.password = ";
    const password = object.content.password;
    if (context.isNew || password) {
        if (password.length < 8) {
            throw "Password is too short. Min length 8 characters";
        }
    }
    return object;
}
```

This code will run before the given object is validated and stored. If this request is a create (`context.isNew` is true) or contains a password, the password is checked to make sure it is long enough. If not, an error is thrown. This error will be returned to the callee and can be displayed as desired.
2.7.9.2 Example: Document Modification

In this slightly more complicated example, we will bind lifecycle hooks to the Document type pre-defined in Cordra with the following features:

- Add a timestamp to the description of the document in a way it is stored.
- Add a timestamp to the description when the object is resolved, but not actually store.
- Require that the description be changed to “DELETEME” before the document can be deleted.

To demonstrate loading JavaScript from an external file, the function to create the timestamp is in a file called util.js. Create a JavaScript Directory (as described above) and upload this file as a payload named util.js.

```javascript
exports.getTimestampString = getTimestampString;

function getTimestampString(isResolution) {
    const currentDate = new Date();
    if (isResolution) {
        return '\nResolved at: ' + currentDate;
    } else {
        return '\nLast saved: ' + currentDate;
    }
}
```

Next, edit the Document type in Cordra and put the following in the JavaScript field.

```javascript
const util = require('util');

exports.beforeSchemaValidation = beforeSchemaValidation;
exports.onObjectResolution = onObjectResolution;
exports.beforeDelete = beforeDelete;

function beforeSchemaValidation(object, context) {
    if (object.content.description !== 'DELETEME') {
        object.content.description += util.getTimestampString(false);
    }
    return object;
}

function onObjectResolution(object, context) {
    object.content.description += util.getTimestampString(true);
    return object;
}

function beforeDelete(object, context) {
    if (object.content.description !== 'DELETEME') {
        throw 'Description must be DELETEME before object can be deleted.';
    }
}
```

(continues on next page)
Finally, create a new document in Cordra. You should see that whenever the document is updated, a new timestamp is appended to the description. If you view the document's JSON, you should see a single resolution timestamp, which changes on every resolution. Finally, if you try to delete the document without changing the description to "DELETEME" you should see an error message.

### 2.7.9.3 Example: Modification of the Indexed Object

It is possible make changes to the object that is indexed such that it differs from the object that is stored. This is achieved by writing a function called `objectForIndexing`.

```javascript
exports.objectForIndexing = objectForIndexing;

function objectForIndexing(object, context) {
    if (object.content.name == "foo") {
        object.content.otherName = "bar";
    }
    return object;
}
```

In this example if the incoming object has a property called `name` with the value `foo`, a new property will be added to the indexed object called `otherName` with the value `bar`. The object that is stored will not contain the new property but you will be able to search for this object via this property with the query `/otherName:bar`.

### 2.7.9.4 Example: Generating ID

Example JavaScript for generating object ids is shown below. Here we generate a random suffix for the handle in base16 and append it to a prefix. By setting `isGenerateIdLoopable` to true, we ask Cordra to repeatedly call this method until a unique id is generated.

```javascript
const cordra = require('cordra');

exports.generateId = generateId;
exports.isGenerateIdLoopable = true;

function generateId(object, context) {
    return "test/" + randomSuffix();
}

function randomSuffix() {
    return Math.random().toString(16).substr(2);
}
```
2.7.9.5 Example: Creating Handle Values

Example JavaScript for creating handle values is shown below. The JavaScript puts a copy of the information from the Cordra object in the Handle record.

```javascript
exports.createHandleValues = createHandleValues;

function createHandleValues(object) {
    const handleValues = [];
    const dataValue = {
        index: 500,
        type: 'CORDRA_OBJECT',
        timestamp: new Date(object.metadata.modifiedOn).toISOString(),
        data: {
            format: 'string',
            value: JSON.stringify(object.content)
        }
    );
    handleValues.push(dataValue);
    return handleValues;
}
```

2.8 Type Methods

Type methods are Cordra’s way of enabling custom operations to be added to the system. And these operations or methods defined in JavaScript can be enabled in the context of a given type; hence the name Type methods. This way to execute JavaScript is in addition to executing JavaScript methods at specific lifecycle points of the digital object management.

2.8.1 Instance Methods

JavaScript can be used to associate and define arbitrary named methods which can return information about an object and/or modify the object. We refer to these as Instance Methods to indicate that these methods can only be invoked by a client on a specific object instance (using its id).

Necessary object locking is performed by Cordra prior to executing these methods.

Suppose you have an instance of an object that has a property called “name”. The below instance method could be used to retrieve that single property:

```javascript
exports.methods = {};
exports.methods.extractNameInstanceMethod = extractNameInstanceMethod;

function extractNameInstanceMethod(object, context) {
    return object.content.name;
}
```
Instance methods are made available to Cordra by assigning them to the object `exports.methods`. The REST API can be invoked using curl. The parameter `objectId` specifies the object to invoke the method on and `method` specifies the method to call.

```bash
```

The POST body of the method call API is available to the method, either as `context.params` (in which case the POST body is interpreted as JSON) or using `context.dirctIo` (see Direct I/O). The output of the method is either JSON returned by the JavaScript function implementing the method, or can be set using `context.directIo`.

See also [Asynchronous Lifecycle Hooks](#) and [Throwing Errors in Schema JavaScript](#) for setting the output using promises or exceptions.

If the method is accessed via GET (or POST with empty body) the input can be given by the URI query parameter "params".

Instance methods can also act as an alternative means of object update. Here is an example to update the name property:

```javascript
exports.methods = {}; exports.methods.updateNameInstanceMethod = updateNameInstanceMethod; function updateNameInstanceMethod(object, context) {   object.content.name = context.params.newName;   return object.content.name; }
```

Request:

```bash
```

Response:

"some name here"

Note that `beforeSchemaValidation` and `beforeSchemaValidationWithId` are not automatically run when updating an object in this manner. (The `beforeSchemaValidation` or `beforeSchemaValidationWithId` code could be called directly by the method code if desired.)
2.8.2 Static Methods

Static methods are not associated with any particular instance of an object, and are useful for only reading information from one or more objects of any type. No object locking is performed by Cordra prior to the execution of these methods.

Since a static method is not associated with an object instance, it only has the single argument called context. When invoking a static method through the REST API, an optional POST body can be supplied which is made available either as JSON under context.params, or using context.directIo (see Direct I/O).

If the method is accessed via GET (or POST with empty body) the input can be given by the URI query parameter "params".

The output of the method is either JSON returned by the JavaScript function implementing the method, or can be set using context.directIo. In the below example, the function echoes back whatever was included in the params along with a timestamp.

Static methods are made available to Cordra by assigning them to the object exports.staticMethods.

```javascript
exports.staticMethods = {};  
exports.staticMethods.exampleStaticMethod = exampleStaticMethod;  
exports.staticMethods["123/abc"] = exampleStaticMethod;

function exampleStaticMethod(context) {  
    const input = context.params;  
    const result = {  
        input: input,  
        timestamp : new Date().getTime()  
    };  
    return result;  
}
```

Request:

curl -k -X POST 'https://localhost:8443/cordra/call?type=Document&method=exampleStaticMethod'  
-H "Accept: application/json"  
-H "Content-type: application/json"  
-d '{"foo":"hello", "bar":"world"}'  
-H 'Authorization: Bearer ACCESS_TOKEN'

Response:

```
{"input":{"foo":"hello","bar":"world"},"timestamp":1532719152687}
```

The example static method shown in the JavaScript above also demonstrates how a method can be given a handle as an identifier as well as a name. Here the method is also exported with the handle "123/abc".
2.8.3 Service-level Static Methods

Static methods can also be created which apply to the entire Cordra service instead of to a specific Type. These methods should be assigned to exports.staticMethods in the “javascript” property of the “design” object.

Service-level static methods can be called

- With special objectId service, /call?objectId=service&method=methodName
- With the DOIP service id, and in particular as DOIP custom operations targeting the service id. See DOIP and Examples.

For backward compatibility service-level static methods can also be called as follows; this is not recommended for new usage.

- With type CordraDesign, /call?type=CordraDesign&method=methodName
- With objectId design, /call?objectId=design&method=methodName

2.8.4 Allowing GET

By default calling a type method requires an HTTP POST. To allow the use of HTTP GET, you can specify a member of the JavaScript function object called “allowGet” to be true:

```javascript
exports.methods = {}; exports.methods.example = function (object, context) {
    // ...
}; exports.methods.example.allowGet = true;
```

2.8.5 Direct I/O

The object context.directIo provides functionality for directly manipulating the input and output of a type method. If the methods for accessing input are used, it is an error to also access context.params. If the methods for accessing output are used, the value returned from the JavaScript function (which otherwise provides the type method output) will be ignored.

The primary use for context.directIo is to allow accessing input or providing output as bytes instead of as JSON.

The following methods are available:

```javascript
context.directIo.getInputAsStream();
context.directIo.getInputAsJavaReader();
context.directIo.getInputAsUint8Array();
context.directIo.getInputAsString();
context.directIo.getInputAsJson();
```

(continues on next page)
context.directIo.getOutputStreamAsJavaOutputStream();
context.directIo.getOutputStreamAsJavaWriter();
context.directIo.getOutputStreamUint8Array(bytes);
context.directIo.getOutputStreamString(string);
context.directIo.getOutputStreamJson(json);

// Access to the Content-Disposition: and Content-Type: request and response headers
context.directIo.getInputFilename();
context.directIo.getInputMediaType();
context.directIo.setOutputFilename(filename);
context.directIo.setOutputMediaType(mediaType);

Note that the writeOutput functions may be called multiple times.

### 2.8.6 Request Attributes

In addition to the "input" available via context.params or context.directIo, type methods can also make use of "request attributes", a JSON object available as context.attributes. In DOIP all requests can optionally carry request attributes as a standard part of the protocol; in the Cordra HTTP REST API, attributes are made available via query parameters on the "call" API.

Consider for example the following JavaScript in the Document schema with the static schema method `workWithAttributes`:

```javascript
exports.staticMethods = {};
exports.staticMethods.workWithAttributes = workWithAttributes;

function workWithAttributes(context) {
    return {
        attributes: context.attributes,
        requestContext: context.requestContext,
        filename: context.directIo.getInputFilename(),
        mediaType: context.directIo.getInputMediaType(),
        input: context.params
    }
}
```

And consider a DOIP request with `targetId` the Document schema object, operationId "workWithAttributes", and "attributes" as follows:

```json
{
    "filename": "testFilename",
    "requestContext": {
        "requestContextKey": "requestContextValue"
    }
}
```
The output from this request is

```
{
  "attributes": {
    "filename": "testFilename",
    "requestContext": {
      "requestContextKey": "requestContextValue"
    },
    "mundaneAttribute": "testMundaneAttribute"
  },
  "requestContext": {
    "requestContextKey": "requestContextValue"
  },
  "filename": "testFilename",
  "input": {
    "inputKey": "inputValue"
  }
}
```

The same can be accomplished using the /call HTTP API with type=Document and method=workWithAttributes.

### 2.9 Java Hooks and Methods

Cordra supports writing lifecycle hooks and type methods entirely in Java. This may be of benefit not simply for language preference reasons, for example if have a need for a multi-threaded background process or if you need to integrate with existing code that is written in Java.

The cordra-hooks-support library can be included in your Gradle or Maven build files.

**Gradle:**

```gradle
compile group: 'net.cnri.cordra', name: 'cordra-hooks-support', version: '2.5.0'
```

**Maven:**

```xml
<dependency>
  <groupId>net.cnri.cordra</groupId>
  <artifactId>cordra-hooks-support</artifactId>
</dependency>
```
2.9.1 Lifecycle Hooks

To implement lifecycle hooks in Java create a new class that implements CordraTypeInterface. This interface contains no-op default methods so you need only override those methods that you wish to implement.

```java
@CordraType("Foo")
public class FooCordraType implements CordraTypeInterface {

    @Override
    public CordraObject beforeSchemaValidation(CordraObject obj, HooksContext context) throws CordraException {
        obj.content.getAsJsonObject().addProperty("name", "added example property");
        return obj;
    }
}
```

Note that this class is annotated with the annotation @CordraType which has an element (optionally called name; if both are specified name takes precedence over value). The @CordraType indicates to Cordra that this class should be loaded and the annotation element indicates which Cordra type to associate it with. In this example if you had a Cordra type called "Foo" any lifecycle hooks implemented in this class would be used for objects of type Foo. The class name FooCordraType is used for the Cordra type name if the annotation element is omitted, and is otherwise irrelevant.

```java
@CordraType("Foo")
public class FooCordraType implements CordraTypeInterface {

    @Override
    public CordraObject beforeSchemaValidation(CordraObject obj, HooksContext context) throws CordraException {
        obj.content.getAsJsonObject().addProperty("name", "added example property");
        return obj;
    }
}
```

The above example demonstrates implementing the beforeSchemaValidation lifecycle hook. Here the hook is adding a new property "name" to the content of the object. The parameter CordraObject obj has the same interface as the Java Cordra client library. See Cordra HTTP REST Client Library - Java Version for examples of using CordraObject. After making changes to the CordraObject, return it to end the hook.

All of the supported lifecycle hooks with their signatures are shown below:

```java
DefaultAcls getAuthConfig(HooksContext context) throws CordraException {
    return null;
}

CordraObject beforeSchemaValidation(CordraObject obj, HooksContext context) throws CordraException {
}
```
CordraObject beforeSchemaValidationWithId(CordraObject obj, HooksContext context) throws CordraException {
    return obj;
}

CordraObject onObjectResolution(CordraObject obj, HooksContext context) throws CordraException {
    return obj;
}

void onPayloadResolution(CordraObject co, HooksContext context) throws CordraException {} 

void objectForIndexing(CordraObject obj, HooksContext context) throws CordraException {
    return obj;
}

void beforeStorage(CordraObject co, HooksContext context) throws CordraException {} 

void afterCreateOrUpdate(CordraObject co, HooksContext context) throws CordraException {} 

void beforeDelete(CordraObject co, HooksContext context) throws CordraException {} 

void afterDelete(CordraObject co, HooksContext context) throws CordraException {} 

void onUnload() throws CordraException {} 

Note that there is an additional onUnload method you can override. This is called when Cordra unloads the class, for example if you update the Java payload containing the jar file with a newer version. You might choose to use it if you need to stop any background processes.

2.9.1.1 Service-Level Lifecycle Hooks

Some lifecycle hooks are at the service level rather than related to a particular type. To implement those in Java create a class the implements CordraServiceHooksInterface and give it the annotation @CordraServiceHooks:

```java
@CordraServiceHooks
public class DesignCordraType implements CordraServiceHooksInterface {
    // Implementation of CordraServiceHooksInterface methods
}
```
All of the supported service-level lifecycle hooks with their signatures are shown below:

```java
default List<HandleValue> createHandleValues(CordraObject co, HooksContext context) {
    return null;
}

default String customizeQuery(String query, HooksContext context) {
    return query;
}

default SearchRequest customizeQueryAndParams(SearchRequest queryAndParams, HooksContext context) {
    return queryAndParams;
}

default AuthenticationResult authenticate(RequestAuthenticationInfo authInfo, HooksContext context) {
    return null;
}
```

If the service-level hooks also implement `CordraTypeInterface`, those hooks will be global defaults that apply to all objects except where the object’s particular type overrides the hook.

### 2.9.2 Type Methods

Type methods, both instance and static, can also be implemented in Java.

```java
@CordraType("Bar")
public class BarCordraType {
    @CordraMethod
    public JsonElement exampleInstanceMethod(CordraObject co, HooksContext context) {
        return new JsonObject();
    }

    @CordraMethod("test/123")
    public JsonElement anotherExampleInstanceMethod(CordraObject co, HooksContext context) {
        return new JsonObject();
    }

    @CordraMethod("echo", "test/xyz")
    public static JsonElement echo(HooksContext context) throws IOException {
        // Method body
    }
}
```
As before create a class and give it the @CordraType annotation with a name that corresponds with the name of the Cordra type object.

To create an instance Cordra method write a method on your class that takes as its arguments a CordraObject and a HooksContext. Then give that method the @CordraMethod annotation. See the method called exampleInstanceMethod in the above example. If you wish to use a different name for the method, for example one that contains characters that are not permitted in Java method names, you can specify an annotation element, optionally called name or names (if multiple annotation elements are specified, names takes precedence over name which takes precedence over value).

To create a static Cordra method write a method on your class and give it the static access modifier. See the method called echo in the above example.

The @CordraMethod annotation has another annotation element called allowGet; if set to true the method can be called using GET in addition to POST. Note that to set both the method names and allowGet you must include names or name.

```
@CordraMethod(names={"echo", "test/xyz"}, allowGet=true)
public static JsonElement echo(HooksContext context) throws IOException {
    JsonElement params = context.getParams();
    return params;
}
```

Static methods can also be implemented at the service level, in the class annotated with @CordraServiceHooks; such methods can be called by targeting the object id "service".

### 2.9.3 Building and Deployment

You should compile and package your java project into a “fat jar”. Which is to say any dependencies you have should be included in the package.

To deploy your jar file you should store it as a payload on an object of type Schema or on the CordraDesign object. The payload must have the name “java”. You need not have a single @CordraType in a single jar nor do you need to store this jar payload on a Schema object with the corresponding type name.

When you save the Schema or CordraDesign object Cordra will, at runtime, find any classes that are appropriately annotated and load them into their own class loader. Which is to say each jar file deployed as a payload is given its own class loader. This approach allows other jars to be loaded in such a way that they do not interfere with each other.
2.10 Handle Integration

Cordra allots identifiers to digital objects. See Identifiers for details on how those identifiers are allotted. In some cases, Cordra may automatically generate the identifier, in which case it will use a prefix configured as discussed below in Handle Minting Configuration.

We describe below three different ways in which those digital object identifiers, aka handles, can be made resolvable via the Digital Object Identifier Resolution Protocol (DO-IRP). See also the DO-IRP v3.0 specification.

2.10.1 DO-IRP Interface

Cordra provides a resolution interface to identifiers. To enable this, edit the handleServerConfig section of the Cordra Design Object. The DO-IRP interface will only be started when the enabled flag in this section is set to true, which is the case by default.

In the handleServerConfig section, set the listenAddress, listenAddresses, tcpPort, externalAddress, and externalTcpPort as needed based on your Cordra and network configuration. externalAddress and externalTcpPort will be how the users connect to the DO-IRP interface; they can be omitted if they match listenAddress and tcpPort. You can set logAccesses to true in order to obtain DO-IRP access logs.

Cordra automatically applies the updated settings once saved; you do not need to restart Cordra to activate the settings.

The DO-IRP requires a public-private key pair to identify the provider (in this case, the Cordra instance) to clients. The keys are managed as files in the Cordra data directory. The standard Cordra key pair (privatekey and publickey) will be used by default. However if you wish to use a different key pair for the DO-IRP interface you can include custom keys handlePrivateKey.bin and handlePublicKey.bin in the data directory.

In order for Cordra to return useful handle records, you should set the Handle Minting Configuration on the Design object, which should have either a baseUri or a javascript property, or a configured Java service-level createHandleValues hook, to define the structure of the handle records.

If you wish to register your DO-IRP interface with an MPA, you or they will need to obtain the DO-IRP interface site information, which is provided by Cordra in response to a get site info request; this can be obtained using the hdl-getsiteinfo script provided in the bin directory of the Cordra distribution.

2.10.2 Handle Server

In certain cases it may be useful to have Cordra rely on external DO-IRP interfaces rather than use its own. For example, your Cordra server may be part of a community that provides a common DO-IRP service. Or you may have handles other than those that Cordra allots and you prefer to manage all those handles using an existing DO-IRP service.

Cordra will register handles in an external service whenever the handleMintingConfig has a handleAdminIdentity configured, along with either a baseUri or javascript property, or a con-
figured Java service-level createHandleValues hook, to define the structure of the handle records. The rest of this section will describe the necessary configuration in more detail.

You can setup a DO-IRP service using the handle server software here.

You will need a Handle key pair to use with Cordra. You can either use the admin keys that were generated during the Handle server installation (admpub.bin and admpriv.bin), or generate a new set of keys using the Handle tools. For example, run the following script from the Handle distribution directory to generate new keys and save them to files called privatekey and publickey:

```bash
./bin/hdl-keygen privatekey publickey
```

del-keygen script is also made available in the bin directory of the Cordra distribution.

You should copy the generated keys to the Cordra data directory and make sure they are named privatekey and publickey. Then, create an admin handle record on your handle server that includes the Cordra public key in the HS_PUBKEY value. That handle will be used as the administrator identity by Cordra when it contacts the handle server. The handle server should be configured so that that identity is authorized to create handle records.

You should edit the Cordra Design Object, specifically the handleMintingConfig section, to set the handleAdminIdentity to be the index and handle of the Cordra admin handle record you created. You will also need to set either baseUri or javascript on the handleMintingConfig as described below, or a configured Java service-level createHandleValues hook, and then save the Design object.

Cordra will automatically register handle records with the external Handle Server for digital objects that are created in the future. To update handle records pertaining to existing digital objects, login into the Cordra UI as admin, select Admin -> Handle Records from the menu, and click on Update All Handles button to have Cordra update the handle records for existing digital objects.

### 2.10.3 Handle Server Storage

This option was relevant when Cordra software did not provide a built-in DO-IRP interface (option one above). Per this option, you can configure an external handle server to use Cordra as its storage.

Like the previous option, you can setup an external handle server using the instructions stated here. You will then have to install the cordra-client-handle-storage software, included with the Cordra distribution, on the handle server. Installation instructions are included in the README.txt file from the cordra-client-handle-storage directory.

### 2.10.4 Handle Minting Configuration

The Design Object includes a section handleMintingConfig which allows configuring both how Cordra automatically generates the identifiers themselves, as well as the values of the handle records.

The prefix property gives the default handle prefix used by Cordra when generating identifiers. If Generate Object Id Hook is used to specify identifiers, or calls to the Cordra APIs always specify the identifier of newly created objects, this will have no effect. Otherwise this should be set to an appropriate prefix.
Other properties specify the values of the handle records. You must set either the `baseUri` property, or have the `createHandleValues` hook defined (in `javascript`, or in the main design `javascript`, or as a Java service-level hook), in order to have Cordra produce handle records during DO-IRP resolution. All these properties are available via the Cordra UI once you login as admin: select `Admin` → `Handle Records` from the menu to access the page to set these two values.

`baseUri` is useful for handle records to return URLs that handle web proxy services (e.g., http://hdl.handle.net) can use to redirect web browsers to the Cordra's HTTP API endpoint or its UI. See the next sub-section for details.

`javascript` is the `Create Handle Values Hook`. An example is here. You can use this hook to return desired handle records beyond just returning URLs. The `createHandleValues` function can also be defined on the main design `javascript` instead of this special `handleMintingConfig` `javascript`, or as a Java service-level hook.

By default, Cordra returns values that are useful for DOIP clients to auto-locate digital objects via the Cordra's DOIP interface. You can turn off Cordra from returning these values by setting the property `omitDoipServiceHandleValue` in the Design Object within the section `handleMintingConfig` to true.

### 2.10.4.1 Setting the Cordra Base URI

By using `baseUri`, handles minted by Cordra will be associated with handle records that consist of a value that handle web proxies can use to redirect the web browser to the Cordra's REST API and/or user interface. The configured base URI will be used for the URLs in the generated handle records.

**Note:** The Cordra base URI must end with a slash.

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 optionally `defaultLinks` and/or `schemaSpecificLinks`. `defaultLinks` is an array of objects indicating which links will be included in the handle records; `schemaSpecificLinks` is an object where each property name is a type, and the corresponding property value is an array of objects indicating links. 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
- `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:

```
{
    "prefix": "20.500.123",
    "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
        }
    ]
}
```

If `defaultLinks` is omitted, Cordra will use a primary `ui` link and a non-primary `json` link.

### 2.11 Digital Object Versioning

This is an experimental feature.

Cordra has support for simple object versioning. Authorized users, i.e., users with write permission, of an object can request Cordra to create a version of an existing object. A complete copy of the original object is made and that copy is 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. The new version also points at the object that it is made a version of.

Any version of an object should be thought of as a snapshot of the current object at a particular point in time. By default version objects may not be edited. If you desire to override this behavior and make versioned objects editable, set the `enableVersionEdits` property on the Design object to `true`.

Cordra UI has controls for creating a version of an object and listing existing versions of an object. Search will only find version objects when the parameter “includeVersions” is set to `true`. 
2.12 Digital Object Hashing

This is an experimental feature.

Cordra has a feature that when enabled automatically generates hashes of an object and its components on the server when an object is created or modified.

Hash generation is configured on a per type basis. To enable hashing for a particular type set the `hashObject` property on the schema object to `true`.

The generated hashes are stored as a JSON object within the object’s `metadata` property. Each hash is a hex-encoded string of the bytes using SHA-256.

```json
{
   "hashes": {
      "alg": "SHA-256",
      "content": "c39e84af6b49c3c6dee72b3d6fa912216f12b8ddb1507f2283ea4a1f5e2f751d",
      "payloads": {
         "payloadName": "c5601d266987ac885ca96522b1b4e439feb7eca39f0fe1111f7342b63b6468f3"
      },
      "full": "8a3251e1a133c65630d056856f0049b3feb82d0bebc5de58b513c09ccb109ad"
   }
}
```

The `content` hash is a hash of the JSON content of the object. The JSON is first canonicalized, ensuring consistent property ordering and value formats.

The bytes of each payload are individually hashed and stored in the `payloads` section of the hashes object.

If the object contains `userMetadata`, the hashes JSON object will contain an additional hash for that component.

The `full` hash is a hash over the full Cordra object including the hashes of any payloads.

It is possible to verify the hashes in server-side JavaScript, such as in lifecycle hooks or Type methods, as follows:

```javascript
const cordraUtil = require('cordra-util');

const verificationReport = cordraUtil.verifyHashes(cordraObject);
```
3.1 Introduction

Cordra provides three different programmable interfaces in addition to a graphical interface for web browsers.

Digital Object Interface Protocol (DOIP) including DOIP API for HTTP Clients, HTTP REST API, and Digital Object Identifier Resolution Protocol (DO-IRP) are the three programmable interfaces. The software distribution includes client libraries, both a Java version and a JavaScript/TypeScript version.

3.1.1 Digital Object Interface Protocol (DOIP) Interface

DOIP is a communication protocol that specifies how clients may interact with digital objects (DOs) that are managed by DOIP services. The method of such interaction is primarily using identifiers associated with digital objects, including those that represent operations, types, and clients.

DOIP is an appropriate choice for users who are interested in an architectural style focused on invoking identified operations, or who focus on persistence or interoperability benefits.

For details about the Cordra’s implementation of DOIP along with examples, see DOIP and Examples. For information about DOIP client library, see DOIP Client Library - Java Version. See also DOIP API for HTTP Clients. See also the DOIP v2 specification.

3.1.2 HTTP REST API

An HTTP API reduces the entry barrier and enables users to leverage Cordra’s features using most HTTP client libraries. We recommend the use of DOIP API for HTTP Clients which allows the use of the DOIP paradigm for clients using HTTP.

Cordra also provides an older RESTful HTTP API for interacting with digital objects, which is still supported.
3.1.3 Digital Object Identifier Resolution Protocol (DO-IRP) Interface

DO-IRP is a rapid resolution protocol for retrieving state information, such as location, public keys, and digests, of a digital object from its identifier.

DO-IRP is originally defined under a different name in RFCs 3650, 3651, and 3652. DO-IRP specification, and its evolution, is currently overseen by the DONA Foundation as part of the Handle System.

Cordra provides an DO-IRP interface enabling clients to rapidly resolve digital object identifiers to their state information. You may access DO-IRP client libraries here.

See Identifiers and Handle Integration for more information about configuring Cordra identifiers and its DO-IRP interface. See also the DO-IRP v3.0 specification.

3.2 DOIP and Examples

The Digital Object Interface Protocol (DOIP) v2 is a specification of the DONA Foundation. Please click here for accessing the specification.

DOIP is a communication protocol that specifies how clients may interact with digital objects (DOs) that are managed by DOIP services. The method of such interaction is primarily using identifiers associated with digital objects, including those that represent operations, types, and clients. In this context, a DOIP service, such as a running Cordra instance, itself is considered a digital object. DOIP is an appropriate choice for users who are interested in an architectural style focused on invoking identified operations, or who focus on persistence or interoperability benefits.

In DOIP, each request represents a client invoking an operation on a target object. The client, operation, and target each have a unique persistent identifier.

This document does not describe further how the DOIP protocol works or the expected structure of digital objects, for which refer to the DOIP v2 specification. Instead this document describes how to invoke DOIP operations on Cordra-managed digital objects using examples.

See DOIP API for HTTP Clients for how to use the DOIP operations model in an HTTP context.

3.2.1 Configuration

DOIP interface can be enabled via configuration in the Design Object. By default, a new Cordra instance will have a DOIP listener on port 9000, on the same listen address as the HTTP/HTTPS ports. This can be turned off or changed by editing the Design object.

Per DOIP, the DOIP service itself is sometimes the appropriate target for an operation. By default, the identifier for the DOIP service is the handle obtained by using the configured prefix from handleMintingConfig (see Handle Integration) together with the suffix “service”. A different serviceld can be configured as follows:

```json
{
   "doip": {
      (continues on next page)
```
This configuration should suffice for most uses. Information about other configuration options follows.

### 3.2.1.1 listenAddress

The primary address to which the DOIP interface should bind. Defaults to the same as the configured listenAddress for HTTP interfaces, which defaults to localhost.

### 3.2.1.2 listenAddresses

An array of IP addresses to which the DOIP interface should bind. If `listenAddress` is also specified, Cordra will listen on all addresses specified.

### 3.2.1.3 port

The TCP port which the DOIP interface should bind to.

### 3.2.1.4 backlog

The backlog of incoming connections for the listening socket. Defaults to 50.

### 3.2.1.5 maxIdleTimeMillis

The maximum time for the server to wait for bytes from a client connection. Defaults to 5 minutes.

### 3.2.1.6 numThreads

The maximum number of server threads for handling DOIP connections. In this experimental setting, defaults to 20.
3.2.1.7 tlsConfig.id

The identity to use for the DOIP TLS certificate. Defaults to the same as processorConfig.serviceId. The TLS keys can be configured via files in the Cordra data directory. By default the standard Cordra key pair files will be used (privatekey and publickey); however, if you wish you can supply custom keys for the doip interface by adding files named doipPrivateKey and doipPublicKey (in Handle format) to the data directory.

3.2.1.8 processorConfig.serviceId

The identifier used for the target of operations intended for the DOIP service itself. Defaults to the configured handleMintingConfig.prefix together with the suffix “service”.

3.2.1.9 processorConfig.serviceName, processorConfig.serviceDescription

Commentary to be returned in the 0.DOIP/Op.Hello response.

3.2.1.10 processorConfig.address

The public address to advertise to clients via the 0.DOIP/Op.Hello operation. Defaults to the listenAddress.

3.2.1.11 processorConfig.port

The public port to advertise to clients via the 0.DOIP/Op.Hello operation. Defaults to the configured port.

3.2.1.12 processorConfig.publicKey

The public key of the DOIP server to advertise to clients via the 0.DOIP/Op.Hello operation. In JWK format. Defaults to the public key used in the certificates.

3.2.2 Cordra Objects serialized for DOIP

A Cordra object has various components:

- id
- type
- content
- acl
- metadata
- payloads
The id and type map directly onto the corresponding properties of a Digital Object as defined in DOIP specification. The content, acl, and metadata properties of a Cordra object become attributes and the payloads become elements of the Digital Object.

3.2.3 DOIP Java Client

In order to instantiate a Java DOIP client, you will need the doip-sdk jar file, along with the dependencies as listed here. You may refer to DOIP Client Library - Java Version for details on how to access the DOIP client library in Java. The jar files are also included in the sw/lib and sw/lib/cordra-client directories of the Cordra distribution.

Instantiate net.cnri.doip.client.DoipClient and call any of its connect methods to create a net.cnri.doip.client.DoipConnection with a DOIP service. The methods of the DoipConnection can then be used to send DOIP requests and receive DOIP responses.

3.2.4 DOIP Examples

A DOIP service listener uses TLS, but many DOIP requests can be entered as plain text within a TLS session. To experiment with this on the command line, you can use for example:

```
openssl s_client -connect localhost:9000
```

Once connected, you can send requests and receive responses.

To briefly recap the DOIP specification, a DOIP message is a sequence of “segments” separated by newline # newline, and terminated by newline # newline # newline. JSON segments contain JSON directly; bytes segments begin with @ newline, followed by multiple “chunks”, each of which is a decimal number indicating a number of bytes, followed by that many bytes, followed by a newline. But many DOIP requests can be sent as JSON followed by two lines each with a # character. Some examples follow.

3.2.4.1 Hello

Request:

```json
{
  "targetId": "20.500.123/service",
  "operationId": "0.DOIP/Op.Hello"
}
#
#
```

Response:

```json
{
  "status": "0.DOIP/Status.1",
  (continues on next page)
3.2.4.2 List Operations

Request:

```json
{
    "targetId": "20.500.123/service",
    "operationId": "0.DOIP/Op.ListOperations"
}
```

Response:

```json
{
    "status": "0.DOIP/Status.1",
}
```
### 3.2.4.3 Search

**Request:**

```json
{
   "targetId": "20.500.123/service",
   "operationId": "0.DOIP/Op.Search",
   "attributes": {
      "query": "+type:Schema +/name:User"
   }
}
```

**Response:**

```json
{"status": "0.DOIP/Status.1"}
{
   "size": 1,
   "results": [
      {
         "id": "test/ccf24d69f39aafee2195",
         "type": "Schema",
         "attributes": {
            "content": {
               "identifier": "test/ccf24d69f39aafee2195",
               "name": "User",
               "schema": {
                  ...
               }
            }
         }
      }
   ]
}
```
3.2.4.4 Create

In general a creation request must specify the “type” and an “attribute” called “content”. If you wish to specify the id of the newly created object, specify an “id” as a sibling property of “type” and “attributes”.

Request:

```
{
    "targetId": "20.500.123/service",
    "operationId": "0.DOIP/Op.Create",
    "input": {
        "type": "User",
        "attributes": {
            "content": {
                "username": "user",
                "password": "password"
            }
        }
    },
    "authentication": {
        "username": "admin",
        "password": "password"
    }
}
```

Response:

```
{
    "status": "0.DOIP/Status.1",
    "output": {
        "id": "test/12dea96fec20593566ab",
        "type": "User",
        "attributes": {
            "content": {
                "id": "test/12dea96fec20593566ab",
                "username": "user",
                "password": ""
            },
            "metadata": {
                "createdOn": 1537467895407,
                "createdBy": "admin",
                "modifiedOn": 1537467895450,
                "modifiedBy": "admin",
                "txnId": 6
            }
        }
    }
}
```

(continues on next page)
Request:

```
{
  "clientId": "test/12dea96fec20593566ab",
  "targetId": "20.500.123/service",
  "operationId": "0.DOIP/Op.Create",
  "authentication": {
    "password": "password"
  }
}
```

Response:

```
DOIP and Examples 96

Hello World
3.2.4.5 Retrieve

Request:

```json
{
    "targetId": "test/0a4d55a8d778e5022fab",
    "operationId": "0.DOIP/Op.Retrieve"
}
```

Response:

```json
{
    "status": "{0.DOIP/Status.1",
    "output":{
        "id":"test/0a4d55a8d778e5022fab",
        "type":"Document",
        "attributes":{
            "content":{
                "id":"test/0a4d55a8d778e5022fab",
                "name":"Hello World"
            },
            "metadata":{
                "createdOn":1537469656224,
                "createdBy":"test/12dea96fec20593566ab",
                "modifiedOn":1537469656235,
                "modifiedBy":"test/12dea96fec20593566ab",
                "txnId":7
            }
        }
    },
    "elements":[
        {
            "id":"file",
            "length":0,
            "type":"text/plain",
            "attributes":{
                "filename":"helloworld.txt"
            }
        }
    ]
}
```
Request:

```json
{
  "targetId": "test/0a4d55a8d778e5022fab",
  "operationId": "0.DOIP/Op.Retrieve",
  "attributes": {
    "element": "file"
  }
}
```

Response:

```json
{
  "status":"0.DOIP/Status.1",
  "output":{
    "id":"test/0a4d55a8d778e5022fab",
    "type":"Document",
    "attributes":{
      "content":{
        "id":"test/0a4d55a8d778e5022fab",
        "name":"Hello World"
      },
      "metadata":{
        "createdOn":1537469656224,
        "createdBy":"test/12dea96fec20593566ab",
        "modifiedOn":1537469656235,
        "modifiedBy":"test/12dea96fec20593566ab",
        "txnId":7
      }
    },
    "elements":[
      {
        "id":"file",
        "length":0,
        "type":"text/plain",
        "attributes":{
          "filename":"helloworld.txt"
        }
      }
    ]
  }
}
```
3.2.4.6 Extended Operations

Cordra enables developers to extend the core functionality and expose that as extended operations. See *Type Methods* for how to write extended operations.

In the case of functionality that is added as an instance Type method, the targetId of the DOIP operation will be digital object identifier to which this instance Type method should be applied. In the case of any static Type method, the targetId will be the digital object identifier of the Type. In either case, the operationId is the identifier that is specified in the JavaScript export statement associated with the method definition.

Below you will find the DOIP request and response structures that applies to the static Type method defined here.

Request:

```json
{
  "targetId": "test/7060f82cc15962ba4851",
  "operationId": "123/abc",
  "authentication": {
    "username": "admin",
    "password": "password"
  },
  "input": {
    "foo": "hello",
    "bar": "world"
  }
}
```

Here test/7060f82cc15962ba4851 is the identifier of the Type digital object on which the static Type method is defined.

Response:

```json
{
  "status": "0.DOIP/Status.001",
  "output": {
    "input": {
      "foo": "hello",
      "bar": "world"
    },
    "timestamp": 1568752848904
  }
}
```

3.3 DOIP API for HTTP Clients

Supported from Cordra v2.3.0.

This section describes the Digital Object Interface Protocol (DOIP) API that HTTP clients can use to interact with Cordra. DOIP API is an RPC-style API designed to be comfortable to develop against, with easy-to-construct URLs and JSON message bodies that convey the essence of DOIP requests and responses. Readers familiar with the DOIP v2 specification may refer to Design Approach to learn how the DOIP message model is mapped to this API.

This documentation is available as an OpenAPI specification, either in HTML or JSON form.

The design of this API is a manifestation of the Digital Object Architecture in which all API requests are seen uniformly as

- a specified client
- performing a specified operation
- on a specified digital object

where a digital object conforms to a specified data model presenting an identifier, a type, and a collection of attributes and elements. The service (Cordra, in this case) managing the digital objects is also considered a digital object.

In all cases, identification of the digital object, operation, service, or client is as a Handle:

- the client identifier is the identifier of its User digital object in the system
- the Cordra service identifier is a fixed identifier (<prefix>/service for purposes of this document)
- the identifiers of the available operations are documented below
- the identifier of the target digital object is the identifier of the relevant object in the system.

3.3.1 Request format

A request can be made with either GET or POST. Only some operations (generally read-only operations) allow GET. Requests target a URL, which is <base URL>/doip. The base URL is constant and will be the domain name (and optionally port) where Cordra is made available (e.g., https://localhost:8443). The examples in this document will use /doip as a shorthand for the target URL.

The query string (after ?) of the URL accepts the following parameters:

- operationId, or o: required, the identifier of the operation to perform; can also be sent in the path after /doip/
- targetId, or t: required, the identifier of the target object
- clientId, or c: the identifier of the caller; this can be implicit in the authentication header sent, in which case it need not be included as a query parameter
- attributes, or a: arbitrary JSON to inform the operation
For convenience, attributes as a JSON object can be expressed as multiple properties with dot separated names. Each such property is combined into the structure of a JSON object. Additionally, query parameters which are not reserved (see Non-reserved Query Parameters) are also combined into the attributes.

For example, all of the following represent the same request:

```
GET /doip?...&attributes={"query":"foo","pageSize":10}
GET /doip?...&a={"query":"foo","pageSize":10}
GET /doip?...&attributes.query=foo&attributes.pageSize=10
GET /doip?...&a.query=foo&a.pageSize=10
GET /doip?...&query=foo&pageSize=10
```

If the value of a query parameter for attributes is valid JSON, then it is parsed as JSON. Otherwise, the value is interpreted as a string.

Finally the “input” to the operation is the POST body, when present. In general it does not matter which Content-Type header is sent, except to avoid certain tools defaulting to the special value application/x-www-form-urlencoded. See next section for more details about this special value.

### 3.3.1.1 POST for long query strings

In certain circumstances (for example, very complicated search queries), it is possible for the query string of the URL to become long enough to potentially be an issue for HTTP servers or middleware. In such a circumstance the same request can be sent using POST with Content-Type: application/x-www-form-urlencoded, as is common in HTTP APIs. Such a POST behaves as if the POST body were appended to the query string of the URL.

For example

```
GET /doip?...&attributes.query=something%20very%20long
```

could be equivalently sent as

```
POST /doip?...
Content-Type: application/x-www-form-urlencoded

attributes.query=something%20very%20long
```

For this reason it is important to set an appropriate Content-Type header for a POST body intended to be the input to an operation; often this will be Content-Type: application/json.
3.3.1.2 Authentication

Authentication can be sent using the standard HTTP Authorization: header, which is translated into the "authentication" property of the DOIP request.

For example, authentication via username and password uses the standard HTTP Basic auth:

```
Authorization: Basic dXNlckBleGFtcGxlLm9yZzpwYXNzd29yZA==
```

and authentication can use an access token (bearer token) acquired using the Access Token operations:

```
Authorization: Bearer 1frevxlceojr3ylc92q2awh3e
```

See below for detail on acquiring access tokens.

3.3.2 Response format

The response will include a header Doip-Response, the value of which is a JSON object, which has a "status" property indicating the DOIP status of the response, and sometimes an "attributes" property with additional data about the response.

The HTTP status code of the response will be set as appropriate for the DOIP status code, and will be one of

<table>
<thead>
<tr>
<th>HTTP Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>The request was processed successfully</td>
</tr>
<tr>
<td>400</td>
<td>There was something wrong with the structure or content of the request</td>
</tr>
<tr>
<td>401</td>
<td>The client must authenticate to perform the attempted operation</td>
</tr>
<tr>
<td>403</td>
<td>The client was not permitted to perform the attempted operation</td>
</tr>
<tr>
<td>404</td>
<td>The requested digital object could not be found</td>
</tr>
<tr>
<td>500</td>
<td>There was an internal server error</td>
</tr>
</tbody>
</table>

The DOIP status code (and corresponding HTTP status code) indicates the nature of the response; responses with DOIP status code other than "0.DOIP/Status.001" (HTTP status code other than 200) are error responses. Error responses will return a response body of `Content-Type: application/json` which is an object with (at minimum) the property "message", a description of the error.

The output of a successful response is contained in the response body. It will often have `Content-Type: application/json`. Details about the structure of the response body is given in the description of each of the operations.
### 3.3.3 Summary of Operations

A summary of various operations along with their identifiers and descriptions is listed below. Operation identifiers that begin with 0.DOIP are defined in DOIP v2 specification. Operation identifiers that begin with 20.DOIP represent Cordra-specific operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.DOIP/Op.Auth.Token</td>
<td>Service</td>
<td>Create a new access token</td>
</tr>
<tr>
<td>20.DOIP/Op.BatchUpload</td>
<td>Service</td>
<td>Batch upload a list of digital objects</td>
</tr>
<tr>
<td>20.DOIP/Op.ReindexBatch</td>
<td>Service</td>
<td>Causes the specified objects to be reindexed</td>
</tr>
<tr>
<td>20.DOIP/Op.Relationships.Get</td>
<td>Object</td>
<td>Get the list of objects related by handles to the target object</td>
</tr>
<tr>
<td>20.DOIP/Op.GetDesign</td>
<td>Service</td>
<td>Get the design and schemas for the system</td>
</tr>
<tr>
<td>20.DOIP/Op.GetInitData</td>
<td>Service</td>
<td>Get a summary of server version, design and current session</td>
</tr>
<tr>
<td>0.DOIP/Op.Hello</td>
<td>Service</td>
<td>Get a hello response from the service</td>
</tr>
<tr>
<td>0.DOIP/Op.ListOperations</td>
<td>Object</td>
<td>Get the list of operations the caller can perform on the target object</td>
</tr>
<tr>
<td>0.DOIP/Op.Create</td>
<td>Service</td>
<td>Create a digital object</td>
</tr>
<tr>
<td>0.DOIP/Op.Retrieve</td>
<td>Object</td>
<td>Retrieve a digital object</td>
</tr>
<tr>
<td>0.DOIP/Op.Update</td>
<td>Object</td>
<td>Update a digital object</td>
</tr>
<tr>
<td>0.DOIP/Op.Delete</td>
<td>Object</td>
<td>Delete a digital object</td>
</tr>
<tr>
<td>0.DOIP/Op.Search</td>
<td>Service</td>
<td>Search for digital objects</td>
</tr>
</tbody>
</table>
3.3.3.1 Aliases

For the convenience of users, short aliases are enabled for referring to the service identifier and to operation identifiers.

In the case of the service identifier, `targetId=service` is considered to be a shortcut for `targetId=<prefix>/service`.

The following table shows operation aliases usable with Cordra.

<table>
<thead>
<tr>
<th>Operation Alias</th>
<th>Operation Handle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>0.DOIP/Op.Create</td>
</tr>
<tr>
<td>Retrieve</td>
<td>0.DOIP/Op.Retrieve</td>
</tr>
<tr>
<td>Update</td>
<td>0.DOIP/Op.Update</td>
</tr>
<tr>
<td>Delete</td>
<td>0.DOIP/Op.Delete</td>
</tr>
</tbody>
</table>

3.3.4 Digital Object JSON Structure

Many of the operations shown below work with the structure of a Digital Object. In general, a Digital Object has a unique resolvable identifier under "id", a "type", further JSON called "attributes", and metadata about "elements" which are (in general) binary payloads associated with the object such as images.

As used with Cordra, "attributes" will always have two top-level properties "content" (the actual type-specific JSON content of the digital object) and "metadata" which is Cordra-maintained metadata such as creation and modification timestamps. Note that depending on the authenticated user, some of the information may be omitted.

For many applications, the most useful part of the structure will be the JSON content of the digital object under "content".

3.3.5 Operation Details

3.3.5.1 Access Token Operations

The access token operations can be used to obtain an access token, check its status, and delete it.

A valid access token can be provided, instead of authentication credentials, for various operations. The system provides an access token only after a successful authentication, and by default the token is valid for 30 minutes from last use. Each valid use renews the lifetime.

Security and performance improvements are usually noted with the use of tokens instead of authentication credentials.
Access token may be sent to operations (other than create access token operation) using an Authorization Bearer header. For example:

```
Authorization: Bearer ACCESS_TOKEN
```

**Create a new access token: 20.DOIP/Op.Auth.Token**

Only POST. Target is the service.

The operation input is a JSON object specifying "grant_type": "password", to indicate that the user is authenticating via username/password, together with the user object identifier (or username) and the password. Note that the HTTPS transport ensures that this is sent encrypted over the network.

The operation output is a JSON object specifying the "access_token" together with certain other informative fields notably "userId".

The properties of the response object:

- `access_token`: The newly created access token.
- `token_type`: Always "Bearer".
- `active`: Whether or not the token is active; always "true" for successful calls of the /auth/token API.
- `username`: Username of the authenticated user
- `userId`: UserId of the authenticated user

Example request:

```
Content-Type: application/json;charset=utf-8
{
    "grant_type": "password",
    "username": "test@example.org",
    "password": "password",
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
{
    "access_token": "14f87409i4ohgjv19bds6wvoy",
    "token_type": "Bearer",
    "active": "true",
    "username": "test@example.org",
}(continues on next page)
```
Check the status of an access token: 20.DOIP/Op.Auth.Introspect

Only POST. Target is the service.

The operation input is a JSON object specifying “token”, the access token to introspect.

The operation output is a JSON object specifying whether the supplied token is “active” and also certain other informative fields notably “userId”.

The properties of the response object:

- active: Whether or not the token is active.
- username: Username of the authenticated user
- userId: UserId of the authenticated user

Example request:

```plaintext
          → full=true
Content-Type: application/json;charset=utf-8

{
  "token": "14f874o9i4ohgjv19bds6wovv"
}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
  "active": "true",
  "username": "test@example.org",
  "userId": "<prefix>/NVSJGF5G"
}
```

Only POST. Target is the service.

Token revocation would typically occur when a user logs out of the system.

Example request:

```plaintext
Content-Type: application/json;charset=utf-8
{
   "token": "14f874o9i4ohgjv19bds6wvov"
}

Response:

HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
{
   "active": "false"
}
```

3.3.5.2 CRUD Operations

Object creation: 0.DOIP/Op.Create

Only POST. Target is the service.

Creation of objects sends operation 0.DOIP/Op.Create to the service object. Regardless of the type of the digital object in the system, the same operation is used. The input should be a Digital Object specifying the “type” to be created and the “content”. The output will be the Digital Object with its new “id” and “metadata” and possibly changes to the “content” automatically populated by the system.

Example request:

```plaintext
POST /doip?operationId=0.DOIP/Op.Create&targetId=service
Content-Type: application/json;charset=utf-8
{
   "type": "Document",
   "attributes": {
      "content": {
         "name": "My Document",
         ...
      }
   }
}
```
Object retrieval: 0.DOIP/Op.Retrieve

Allows GET or POST. Target is the specific digital object to be retrieved.

Retrieval of objects from the system is done using the operation 0.DOIP/Op.Retrieve. Regardless of the type of the digital object in the system, the same operation is used. Retrieval returns a JSON response with a Digital Object structure, which includes the identifier, the type, system-maintained metadata, plus of course the type-specific content of the object.

An element (instead of object metadata) can be retrieved using the request attribute attributes.element=elementName.

Example request:

```
GET /doip?operationId=0.DOIP/Op.Retrieve&targetId=<prefix>/MZ6W9D3T
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
    "id": "<prefix>/MZ6W9D3T",
    "type": "Document",
    "attributes": {
        "content": {
            "id": "<prefix>/MZ6W9D3T",
            "name": "My Document",
            ...
        },
        "metadata": {
            "createdOn": 1607709592349,
            "createdBy": "admin",
            "modifiedOn": 1607709906109,
            "modifiedBy": "admin"
        }
    }
}
```
Object update: 0.DOIP/Op.Update

Only POST. Target is the digital object to be updated.

To update an object send operation 0.DOIP/Op.Update with the target the object to be updated. Regardless of the type of the digital object in the system, the same operation is used. The input should be a Digital Object specifying the new “content” (keeping the “id”, “type”, and other properties not in the content is fine but optional) as a complete replacement. The output will be the Digital Object with its new “metadata” and possibly changes to the “content” automatically populated by the system.

Example request:

POST /doip?operationId=0.DOIP/Op.Update&targetId=<prefix>/MZ6W9D3T
Content-Type: application/json;charset=utf-8

{
  "attributes": {
    "content": {
      "name": "My Document",
      "description": "Updated description",
      ...
    }
  }
}

Response:
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
  "id": "<prefix>/MZ6W9D3T",
  "type": "Document",
  "attributes": {
    "content": {
      "id": "<prefix>/MZ6W9D3T",
      "name": "My Document",
      "description": "Updated description",
      ...
    },
    "metadata": {
      "createdOn": 1607709592349,
      "createdBy": "admin",
      "modifiedOn": 1607899161123,
      "modifiedBy": "admin"
    }
  }
}

Object deletion: 0.DOIP/Op.Delete

Only POST. Target is the digital object to be deleted.
To delete an object send operation 0.DOIP/Op.Delete with, as the targetId, the id of the object to be deleted. Regardless of the type of the digital object in the system, the same operation is used. The input and output (in the case of success) are empty.

Example request:

POST /doip?operationId=0.DOIP/Op.Delete&targetId=<prefix>/MZ6W9D3T

Response:

HTTP/1.1 200 OK
3.3.5.3 Search: 0.DOIP/Op.Search

Allows GET or POST. Target is the service.

Request attributes (send as query parameters, or in a POST body of Content-Type: application/x-www-form-urlencoded, or in a POST body of Content-Type: application/json):

- "query": the search to be performed, in Lucene syntax; see below
- "pageNum": the page number to be returned, starting with 0
- "pageSize": the page size to be returned; if missing or negative, all results will be returned; if zero, no results are returned, but the "size" and, if requested, the "facets" are still returned
- "sortFields": a comma-separated list of sort specifications, each of which is a field name optionally followed by ASC or DESC
- "type": either "id", to return just object ids, or "full", to return full object data; defaults to "full"
- "facets": a JSON array of objects that contain the fields to facet results by
- "filterQueries": a JSON array of query strings to filter the results by
- "includeScore": a boolean which if true will cause the results to include attributes "responseContext" with "score" properties, populated by the index service
- "includeVersions": a boolean which if true will cause the results to include Cordra version objects which (see Digital Object Versioning)

Output is a JSON object with top-level properties:

- "size": the number of results across all pages
- "facets": only included if "facets" are specified in the request. A list of counts by bucket for each facet in the request. Each bucket for a facet also includes a "filterQuery" that can be sent in a subsequent request under "filterQueries" to further restrict the results
- "results": a list of results, each of which is either a string (the object id) or the JSON structure of a digital object

The query syntax is the Lucene syntax used by Lucene, Solr, and Elasticsearch. Here are three versions of the equivalent documentation from the three providers:


This syntax supports fielded search, where the fields use JSON Pointer (RFC 6901) format to specify locations in the structure of the object "content" (but the field names for JSON Pointers into arrays have particular indices replaced with underscore _ in order to allow searching the entire array).

For more information and detailed examples about the query syntax see Query Syntax.
Example request:


Response:

HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8

{
  "size": 123,
  "results": [
    {
      "id": "<prefix>/MZ6W9D3T",
      "type": "Document",
      "attributes": {
        "content": {
          "id": "<prefix>/MZ6W9D3T",
          "name": "My Document",
          ...
        },
        "metadata": {
          ...
        }
      }
    },
    ...
  ]
}

3.3.5.4 Cordra Specific Operations


Request attributes:

- "format": optional, may take the value "ndjson" to send objects as newline separated JSON objects. If omitted the request body should either be an array of Digital Objects or an object containing a "results" property that contains an array of Digital Objects. The three input formats are shown below:

New line separated objects:

```json
{}
{}
{}
```
Array of objects:

```
[
    {},
    {},
    {}
]
```

An object containing a results array:

```
{
    "results": [
        {},
        {},
    ]
}
```

- "pageNum": the page number to be returned, starting with 0

If an object in the request contains an id the operation will either update the corresponding object or create a new one with the specified id. Whereas if an object in the request does not contain an id a create will be performed by the server assigning a new id to the object.

Example request:

```
Content-Type: application/json;charset=utf-8
```

POST Data:

```
[
    {
        "type": "Foo",
        "attributes": {
            "content": {}
        }
    },
    {
        "id": "test/e4673f57012b544e72af",
        "type": "Foo",
        "attributes": {
            "content": {}
        }
    }
]
```

Response:
The data of an element can be included in the json as a base64 string:

Example request:

```
Content-Type: application/json;charset=utf-8
```

POST Data:
The special attribute "elementsToDelete" can be set on the object to list the elements that should be deleted from an existing object.

Example request:

```
Content-Type: application/json;charset=utf-8

POST Data:

```
```json
[
  {
    "id": "test/e4673f57012b544e72af",
    "type": "Foo",
    "attributes": {
      "content": {},
      "elementsToDelete": [
        "element1"
      ]
    }
  }
]
```

Given an array of object ids in the POST body the server will retrieve those objects from storage and caused them to be reindexed by the indexing service.

Request attributes:

- "lockObjects": optional boolean, defaults to true. If true the ids from the batch of objects are locked while being reindexed.
- "query": optional string. Instead of supplying a list of ids to reindex a query can be sent instead. The query will be run server side and the results will be reindexed.
- "all": optional boolean. Instead of supplying a list of ids to reindex, all stored objects will be reindexed.

Example request:

```bash
Content-Type: application/json;charset=utf-8

POST Data:

[
  "test/bc8e231bb247ce2a2a76",
  "test/dd36b5b638e97e6c608f"
]
```


Request attributes:

- "outboundOnly": optional boolean, defaults to false. By default the response will show objects that the target object points at as well as objects that point at the target. Setting this attribute to true restricts the results to only those the target points at.

Example request:

```bash

Response:

```
{
  "id": "test/55de0539eb1e14f26a04"
},
"edges": [
  {
    "from": "test/d39366ae3f76b6f41384",
    "to": "test/55de0539eb1e14f26a04",
    "style": "arrow",
    "jsonPointer": "/ACTED_IN/0"
  }
],
"results": {
  "test/55de0539eb1e14f26a04": {
    "id": "test/55de0539eb1e14f26a04",
    "type": "Movie",
    "attributes": {
      "content": {
        "title": "Top Gun",
        "released": "1986"
      },
      "metadata": {
        "createdOn": 1655743375214,
        "createdBy": "admin",
        "modifiedOn": 1656429032809,
        "modifiedBy": "admin",
        "txnId": 1656429032800092
      }
    }
  },
  "test/d39366ae3f76b6f41384": {
    "id": "test/d39366ae3f76b6f41384",
    "type": "Person",
    "attributes": {
      "content": {
        "name": "Tom Cruise",
        "born": "1964",
        "ACTED_IN": [
          "test/55de0539eb1e14f26a04"
        ]
      },
      "metadata": {
        "createdOn": 1655743424735,
        "createdBy": "admin",
        "modifiedOn": 1657913699440,
        "modifiedBy": "admin"
      }
    }
  }
}

[DOIP API for HTTP Clients](#)

Request attributes:

- "clonePayloads": optional boolean, defaults to true. If true the bytes of the objects elements will be copied into the new version object.
- "versionId": optional string. If supplied this will be the id for the new version object. If omitted a

Example request:

```
```

Response:

```
{
    "id": "test/63fcaf42ecde53b68b10",
    "type": "Foo",
    "versionOf": "test/e4673f57012b544e72af",
    "publishedBy": "admin",
    "publishedOn": 1657808342642,
    "isTip": false
}
```


Example request:

```
```

Response:

```
[
    {
        "id": "test/63fcaf42ecde53b68b10",
        "type": "Foo",
        "versionOf": "test/e4673f57012b544e72af",
        "publishedBy": "admin",
        "publishedOn": 1657808342642,
        "isTip": false
    }
] (continues on next page)
3.3.6 Design Approach

This section describes the design approach followed to map messages that conform to DOIP v2 specification to requests and responses as used by this DOIP API. Readers of this appendix are expected to be familiar with the DOIP v2 specification.

3.3.6.1 Request Mapping

A request to this DOIP API can be made with either GET or POST, which gets mapped to request form defined in the DOIP v2 specification. Only some DOIP operations (generally read-only operations) allow GET; this is part of the specification of the operation.

The examples in this document will use /doip, but the full URL should be constructed using the endpoint of the Cordra service. For example, this could be a URL like https://domain.name/doip or https://domain.name/cordra/doip depending on the details of the Cordra installation.

The components of a DOIP request (section 7.2.1 of the DOIP v2 specification) are obtained from the HTTP request as follows. Each query parameter has both a long form and a short single-character form; multiple values is an error for parameters representing a string (requestId, operationId, targetId, clientId), and multiple values will be combined for parameters representing JSON values (authentication, attributes, input).

- requestId: from the query parameter “requestId” or “r”. Can be omitted as the HTTP transport provides its own mechanisms for associating responses to requests.
- operationId: from the query parameter “operationId” or “o”. If no operationId is given by the query parameters, the URI path after /doip with the initial slash omitted is used as the operationId.
- targetId: from the query parameter “targetId” or “t”, required
- clientId: from the query parameter “clientId” or “c”; can be omitted if the authentication provides it implicitly
Authentication can be sent using the standard HTTP Authorization: header, which is translated into the "authentication" property of the DOIP request.

Authorization: Basic is translated into an "authentication" object with "username" and "password" properties.

Authorization: Bearer is translated into an "authentication" object with a "token" property.

For Authorization: Doip, the rest of the Authorization: header is Base64-decoded and parsed as JSON to provide an arbitrary "authentication" property for the request.

If an Authorization: header is present and an "authentication" or "u" query parameter is present, the properties from the query parameter are added to the object generated from the Authorization: header, with repeated properties treated as an error.

JSON Query Parameters

For convenience, properties of the DOIP request which represent JSON can be expressed as multiple properties in the API request with dot separated names. This is done for "attributes", "authentication", and "input" along with their short names "a", "u", and "i".

Taking "attributes" as an example, if the value of some attributes.prop is valid JSON, then it is parsed as JSON. Otherwise, the value is interpreted as a string. For example,

GET /doip?...&attributes={"query":"foo","pageSize":10}

is exactly the same as

GET /doip?...&attributes.query=foo&attributes.pageSize=10

Any further dot . in the query parameter is interpreted as indicating a deeper subobject; for example attributes.foo.bar=true corresponds to the attributes {"foo":{"bar":true}}.
Values from the long name of the query parameter and the corresponding short name are combined. Multiple properties with the same name is an error, and so is indicating properties of a JSON value which does not represent a JSON object at all.

For authentication, the values from the query parameters are combined with any JSON object generated from the Authorization: header. For input, the values from the query parameters will be combined with any JSON POST body.

Non-reserved Query Parameters

The query parameters “requestId”, “operationId”, “targetId”, “clientId”, “authentication”, “attributes”, and “input” are reserved; every single-character query parameter is reserved; and any query parameter which starts with a reserved query parameter followed by a dot is reserved.

Every non-reserved query parameter will become part of “attributes”, following the same rules for dots as described in JSON Query Parameters. For example

GET /doip?...&query=foo&pageSize=10

represents the attributes {"query":"foo","pageSize":10}; and foo.bar=true represents the attributes {"foo":{"bar":true}}.

Attributes from headers

For convenience, two standard HTTP headers map to specific DOIP request attributes. The Content-Type: header from the API request is copied into an property “mediaType” of the attributes object, if the attributes do not already have a mediaType. The filename from a Content-Disposition header is copied into a property “filename” of the attributes object, if the attributes do not already have a filename.

POST for long query strings

See POST for long query strings above.

Input

For a GET or a POST with empty body, the input property of the corresponding DOIP request is assumed to be empty.

If the Content-Type: of the HTTP request is application/x-www-form-urlencoded, the input will be assumed empty and the POST body is used to populate the query string parameters.

If the Content-Type: of the HTTP request is application/json or ends in +json, the entire body represents a DOIP input which is a single JSON segment.

If the Content-Type: of the HTTP request is multipart/mixed, the body represents a multi-segment DOIP input. The content of each segment is equivalent to the body of the corresponding part, and...
each segment is JSON or bytes depending on whether the Content-Type of the corresponding part is application/json or ends in +json.

If the Content-Type: of the HTTP request is multipart/form-data, the body represents a multi-
segment DOIP input, with additional JSON parts corresponding to the names of every part except the first. This allows a particularly compact representation of the DigitalObject serialization as de-
defined in the DOIP v2 specification. In particular, every part of the multipart request body except the first corresponds to two segments of the DOIP request; the first of those two segments is a JSON segment of the form:

```
{"id":"name-of-the-part"}
```

where name-of-the-part comes from the name parameter of the part's Content-Disposition: header:

```
Content-Disposition: form-data; name="name-of-the-part"
```

The second of the two segments corresponding to each part after the first, and the only segment corresponding to the first part, has content equivalent to the body of the part; each such segment is JSON or bytes depending on whether the Content-Type of the corresponding part is application/
json or ends in +json.

If the Content-Type: is the HTTP request is neither JSON nor multipart, the entire body represents a DOIP input which is a single bytes segment.

### 3.3.6.2 Response Mapping

The components of a DOIP response (section 7.2.2 of the DOIPv2 specification) are mapped into an API response as follows.

The DOIP response "requestId", "status", and "attributes" are mapped into the HTTP response header Doip-Response, the value of which is a JSON object serialized using only ASCII for maxi-
mum HTTP compatibility.

The HTTP status code of the response will be set according to the DOIP status code, and (for basic DOIP status codes) will be one of

<table>
<thead>
<tr>
<th>Status Code</th>
<th>DOIP Status Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.DOIP/Status.001</td>
<td>The request was processed successfully</td>
</tr>
<tr>
<td>400</td>
<td>0.DOIP/Status.101, 0.DOIP/Status.200</td>
<td>There was something wrong with the structure or content of the request</td>
</tr>
<tr>
<td>401</td>
<td>0.DOIP/Status.102</td>
<td>The client must authenticate to perform the attempted operation</td>
</tr>
<tr>
<td>403</td>
<td>0.DOIP/Status.103</td>
<td>The client was not permitted to perform the attempted operation</td>
</tr>
<tr>
<td>404</td>
<td>0.DOIP/Status.104</td>
<td>The requested digital object could not be found</td>
</tr>
<tr>
<td>409</td>
<td>0.DOIP/Status.105</td>
<td>There was a conflict preventing the request from being executed</td>
</tr>
<tr>
<td>500</td>
<td>0.DOIP/Status.500</td>
<td>There was an internal server error</td>
</tr>
</tbody>
</table>
Custom DOIP statuses need to specify the expected corresponding HTTP status code; otherwise 200 is used as a default.

Headers from attributes

For convenience, two standard HTTP headers are populated in the API response according to specific DOIP response attributes. A property “mediaType” of the response attributes is copied into the Content-Type: header. A property “filename” of the response attributes is copied into the Content-Disposition: header.

Output

The DOIP response “output” is contained in the API response body. A JSON segment is written with Content-Type: application/json. Multiple segments are written with a multipart Content-Type. Other Content-Type values indicate a single bytes segment.

3.4 HTTP REST API and Examples

Note: our current recommendation for clients who wish to interact with Cordra using HTTP is DOIP API for HTTP Clients. The older Cordra HTTP API described here is still supported and this documentation can still serve as a reference for certain Cordra behaviors independent of the particular API used to access Cordra.

Cordra provides a RESTful HTTP API for interacting with digital objects. Cordra also provides a web browser based interface that dynamically creates object editors based on configured Schemas/Types, mainly for administrative purposes.

Cordra HTTP requests that conform to HTTP/1.1 and HTTP/2 specifications (RFCs 7230, 7231, and 7540) are tested.

3.4.1 Overview

Main APIs
### Resource

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /objects/&lt;id&gt;</td>
<td>Retrieves an object or part of an object by id.</td>
</tr>
<tr>
<td>POST /objects/?type=&lt;type&gt;</td>
<td>Create an object by type.</td>
</tr>
<tr>
<td>PUT /objects/&lt;id&gt;</td>
<td>Update an object by id.</td>
</tr>
<tr>
<td>GET /search</td>
<td>Search for objects (requires Cordra v2.1.0).</td>
</tr>
<tr>
<td>POST /search</td>
<td>Search for objects (requires Cordra v2.1.0).</td>
</tr>
<tr>
<td>GET /objects/?query=&lt;query&gt;</td>
<td>Search for objects.</td>
</tr>
<tr>
<td>DELETE /objects/&lt;id&gt;</td>
<td>Delete an object by id.</td>
</tr>
<tr>
<td>DELETE /objects/&lt;id&gt;?payload=&lt;payload&gt;</td>
<td>Delete a payload by id and name.</td>
</tr>
<tr>
<td>GET <a href="http://hdl.handle.net/">http://hdl.handle.net/</a>&lt;id&gt;</td>
<td>Retrieves an object via the Handle proxy.</td>
</tr>
<tr>
<td>GET /acls/&lt;id&gt;</td>
<td>Retrieve the ACLs for a specific object.</td>
</tr>
<tr>
<td>PUT /acls/&lt;id&gt;</td>
<td>Modify the ACLs for a specific object.</td>
</tr>
<tr>
<td>PUT /users/this/password</td>
<td>Change the password of the currently authenticated user.</td>
</tr>
<tr>
<td>GET /check-credentials</td>
<td>Retrieve information about provided credentials.</td>
</tr>
<tr>
<td>POST /call</td>
<td>Call a Type method. (Some Type methods may allow the use of GET rather than POST.)</td>
</tr>
<tr>
<td>GET /listMethods</td>
<td>List Type methods.</td>
</tr>
<tr>
<td>POST /batchUpload</td>
<td>Batch upload objects.</td>
</tr>
</tbody>
</table>

### Access Token APIs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST /auth/token</td>
<td>Create a new access token.</td>
</tr>
<tr>
<td>POST /auth/introspect</td>
<td>Retrieve information about the specified access token.</td>
</tr>
<tr>
<td>POST /auth/revoke</td>
<td>Delete specified access token (used to “sign out”).</td>
</tr>
</tbody>
</table>

### Administrative APIs
<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /schemas/&lt;type&gt;</td>
<td>Get schema or all schemas.</td>
</tr>
<tr>
<td>PUT /schemas/&lt;type&gt;</td>
<td>Create or update a schema.</td>
</tr>
<tr>
<td>DELETE /schemas/&lt;type&gt;</td>
<td>Delete a schema.</td>
</tr>
<tr>
<td>POST /uploadObjects</td>
<td>Bulk upload of objects from a json file.</td>
</tr>
<tr>
<td>PUT /adminPassword</td>
<td>Change the password for the admin user.</td>
</tr>
<tr>
<td>POST /updateHandles</td>
<td>Update all handle records for objects in Cordra.</td>
</tr>
<tr>
<td>GET /updateHandles</td>
<td>Get status of handle update.</td>
</tr>
<tr>
<td>POST /reindexBatch</td>
<td>Reindex all specified objects.</td>
</tr>
<tr>
<td>GET /startupStatus</td>
<td>Accessible even after failed startup to indicate success or failure</td>
</tr>
</tbody>
</table>

Experimental APIs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /versions/?objectId=&lt;objectId&gt;</td>
<td>Get version information for a specific object.</td>
</tr>
<tr>
<td>POST /versions/?objectId=&lt;objectId&gt;</td>
<td>Create a version of a specific object.</td>
</tr>
</tbody>
</table>

3.4.2 HTTP Response Codes

<table>
<thead>
<tr>
<th>HTTP Response Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>The request was successfully processed.</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>There was an error with the request, such as a creation or update which is not schema-valid. Actually “Unauthenticated”. There is no authenticated user, or authentication failed, but the requested operation requires authentication.</td>
</tr>
<tr>
<td>401 Unauthorized</td>
<td>The authenticated user does not have permissions to perform the requested operation.</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>The Cordra object or requested part does not exist.</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>An attempt was made to create a Cordra object with a handle already in use by a Cordra object.</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>An unexpected server error occurred. Usually diagnosis requires looking at the server logs.</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>An unexpected server error occurred. Usually diagnosis requires looking at the server logs.</td>
</tr>
</tbody>
</table>

Range requests for payloads can additionally return 206 Partial Content and 416 Range Not Satisfiable, following the standard specification for HTTP range requests.
3.4.3 Authentication

For any Cordra API call that requires authentication, you can either authenticate directly for the single call, or provide an access token. Access tokens are used to maintain a server session in between requests. This way, the server does not need to reprocess your authentication credentials with each request. Using sessions can significantly speed up the process of making multiple requests.

There are two ways to authenticate directly. One is to include username (or, user object id) and password following the HTTP Basic Authentication method. Another is to authenticate using public-private key pair following the HTTP Bearer Authentication method with a JWT. For more information on the topic, see Authentication and Access Controls.

To obtain an access token, you must authenticate using the Access Token API. Once authenticated, you will receive an access token that should be sent back with subsequent calls. This token should be sent using an Authorization Bearer header. For example:

```
Authorization: Bearer ACCESS_TOKEN
```

3.4.4 As-User

The As-User header can be set to allow the admin user to perform operations on behalf of another user. If request comes with proper authentication for admin, and the As-User header is set, that userId will be considered to have authenticated instead.

The Authenticate Hook hook can optionally be used to allow broader non-admin access to As-User functionality.

3.4.5 Request Context

All requests allow a query parameter called requestContext. This must be valid JSON object (suitably encoded as a query parameter or, when building call requests, as an attribute). The requestContext is made available to JavaScript methods and lifecycle hooks as part of the context argument (see Type Methods and Lifecycle Hooks). It is also made available to the StorageChooser used with Multiple Storages.

3.4.6 API 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.

Example Schema:

```json
{
    "type": "object",
    "title": "Document",
    (continues on next page)
```
"required": [
  "name",
  "description"
],
"properties": {
  "id": {
    "type": "string",
    "cordra": {
      "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"
      }
    }
  }
}
3.4.6.1 Objects API

Retrieve object by id

Request:

GET /objects/<id>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>required</td>
<td>The id of the desired object.</td>
</tr>
<tr>
<td>jsonPointer</td>
<td>optional</td>
<td>The jsonPointer into the subcomponent of the target object</td>
</tr>
<tr>
<td>filter</td>
<td>optional</td>
<td>A JSON array of jsonPointers used to restrict the result object.</td>
</tr>
<tr>
<td>payload</td>
<td>optional</td>
<td>The name of the payload to retrieve</td>
</tr>
<tr>
<td>pretty</td>
<td>optional</td>
<td>Format returned json</td>
</tr>
<tr>
<td>text</td>
<td>optional</td>
<td>When present on a request which would normally result in a JSON string, the response is the contents of the JSON string</td>
</tr>
<tr>
<td>disposition</td>
<td>optional</td>
<td>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</td>
</tr>
<tr>
<td>full</td>
<td>optional</td>
<td>If present the response is the full Cordra object, including properties id, type, content, acl, metadata, and payloads. By default only the content is returned.</td>
</tr>
</tbody>
</table>

Request headers

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

<table>
<thead>
<tr>
<th>X-Schema</th>
<th>Type of the object</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Permission</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Examples

**Request:**

GET /objects/20.5000.1234/1321d2d033b22bee1187

**Response:**

```
{
    "id": "20.5000.1234/1321d2d033b22bee1187",
    "name": "A file",
    "description": "It's a file",
    "creator": {
        "fullName": "John Doe",
        "organization": "Acme Corp."
    }
}
```

**Request:**

GET /objects/20.5000.1234/1321d2d033b22bee1187?jsonPointer=/creator

**Response:**

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

**Request:**

GET /objects/20.5000.1234/1321d2d033b22bee1187?jsonPointer=/description&text

**Response:**

It's a file

**Request:**
GET /objects/20.5000.1234/1321d2d033b22bee1187?payload=file

Response:

(Contents of the payload)

To retrieve the Cordra design object, retrieve the object with id “design”.

Create object by type

Request:

POST /objects/?type=Document

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>required</td>
<td>The type of the object being created. In this case “Document”.</td>
</tr>
<tr>
<td>dryRun</td>
<td>optional</td>
<td>Do not actually create the object. Will return results as if object had been created.</td>
</tr>
<tr>
<td>suffix</td>
<td>optional</td>
<td>The suffix of the handle used to identify this object. One will be generated if neither ‘suffix’ nor ‘handle’ is specified.</td>
</tr>
<tr>
<td>handle</td>
<td>optional</td>
<td>The handle used to identify this object. One will be generated if neither ‘suffix’ nor ‘handle’ is specified.</td>
</tr>
<tr>
<td>full</td>
<td>optional</td>
<td>If present the response is the full Cordra object, including properties id, type, content, acl, metadata, and payloads. By default only the content is returned.</td>
</tr>
</tbody>
</table>

See *Handle Minting Configuration* for configuring the handle prefix for automatic handle generation when the “handle” parameter is not used.

**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 |
Examples

POST Data:

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

Response:

```json
{
    "id": "20.5000.1234/23bdf2a62a83225a1b77",
    "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 content which is the JSON content of the object to be created. There may optionally be a part named acl which will be the acl component of the new object; it must be a JSON object with two properties "readers" and "writers", each a JSON array of strings.

Parts which have filenames determine payloads. The payload name is the part name. The filename and a Content-Type if present are stored as the metadata of the payload.

POST Data:

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

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

(continues on next page)
Update object by id

Request:

PUT /objects/<id>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>required</td>
<td>The id of the object to update.</td>
</tr>
<tr>
<td>dryRun</td>
<td>optional</td>
<td>Do not actually update the object. Will return results as if object had been updated.</td>
</tr>
<tr>
<td>type</td>
<td>optional</td>
<td>If specified, indicates a request to change the type of the object. Note that schemas and lifecycle methods will change with the type; in many cases deleting and recreating an object may be preferable.</td>
</tr>
<tr>
<td>payloadToDelete</td>
<td>optional</td>
<td>The name of an existing payload to delete. Can be used multiple times.</td>
</tr>
<tr>
<td>jsonPointer</td>
<td>optional</td>
<td>A JSON pointer within the object’s content. Only the JSON at that JSON pointer will be updated. Note that only the content can be modified, not the full object.</td>
</tr>
<tr>
<td>full</td>
<td>optional</td>
<td>If present the response is the full Cordra object, including properties id, type, content, acl, metadata, and payloads. By default only the content is returned.</td>
</tr>
</tbody>
</table>

Request Headers

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

HTTP REST API and Examples 132
Examples

Request:

PUT /objects/20.5000.1234/23bdf2a62a83225a1b77

PUT Data:

```
{
   "id": "20.5000.1234/23bdf2a62a83225a1b77",
   "name": "A different file",
   "description": "I've changed the description",
   "creator": {
      "fullName": "Jane Doe",
      "organization": "Acme Labs."
   }
}
```

Response:

```
{
   "id": "20.5000.1234/23bdf2a62a83225a1b77",
   "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 name as the value of a `payloadToDelete` parameter. Multiple `payloadToDelete` parameters are allowed.

Search for objects

Cordra v2.1.0 includes a separate /search API. For compatibility with earlier versions of Cordra, the GET /objects/?query=<query> API (with a query and no object ID) should be used.

The /search API supports POST of a JSON body defining the search request. The /objects API only supports GET for searching, except POST with `Content-Type: application/x-www-form-urlencoded` can be used to avoid URI length limitations.

Request:
### Parameters (or properties of application/json POST body)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>optional</td>
<td>The query to be processed. Either query or queryJson is required.</td>
</tr>
<tr>
<td>queryJson</td>
<td>optional</td>
<td>JSON which is used to generate a query for all objects with content matching the supplied JSON. Either query or queryJson is required.</td>
</tr>
<tr>
<td>ids</td>
<td>optional</td>
<td>If specified, the search returns the ids of the matched objects only.</td>
</tr>
<tr>
<td>pageNum</td>
<td>optional, default 0</td>
<td>The desired results page number. 0 is the first page.</td>
</tr>
<tr>
<td>pageSize</td>
<td>optional</td>
<td>The number of results per page. If omitted or negative no limit. If 0 no results are returned, only the size (number of hits).</td>
</tr>
<tr>
<td>sortFields</td>
<td>optional</td>
<td>Sort fields for the query results. A JSON array of objects with properties “name”, a JSON pointer, and optionally “reverse”, a boolean. (A legacy format is comma-separated, with each field name optionally followed by ASC or DESC to indicate the sort direction.)</td>
</tr>
<tr>
<td>full</td>
<td>optional</td>
<td>If set to false only the content of the object is returned.</td>
</tr>
<tr>
<td>filter</td>
<td>optional</td>
<td>A JSON array of jsonPointers used to restrict the result object. The jsonPointers are relative to the root of the objects in the results. Note that that root changes if the full param is set to false.</td>
</tr>
<tr>
<td>filterQueries</td>
<td>optional</td>
<td>A JSON array of JSON strings, each of which is interpreted as a query string specifying additional constraints. Each item in filterQueries is conceptually added to the main query as with a boolean AND; however filterQueries do not affect document scoring, and in some cases may enhance backend performance by allowing caching.</td>
</tr>
<tr>
<td>facets</td>
<td>optional</td>
<td>A JSON array of JSON objects. Each facet object must include a property “field”, value a string, which is an index field on which faceting is requested; the facet object may optionally include a property “maxBuckets”, the maximum number of buckets to return, which defaults to 10; see Faceted Search below.</td>
</tr>
<tr>
<td>includeScore</td>
<td>optional</td>
<td>A boolean; if true, the “score” from the index service will be included in each Cordra object returned, as a property of “responseContext”.</td>
</tr>
<tr>
<td>includeVersions</td>
<td>optional</td>
<td>A boolean; if true, Cordra version objects will be included in the results (see Digital Object Versioning)</td>
</tr>
</tbody>
</table>
Examples

Request:

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

Response:

```
{
   "size": 1,
   "pageNum": 0,
   "pageSize": 10,
   "results": [
      {
         "id": "20.5000.1234/1321d2d033b22bee1187",
         "type": "Document",
         "content": {
            "id": "20.5000.1234/1321d2d033b22bee1187",
            "name": "A file",
            "description": "Its a file",
            "file": "",
            "creator": {
               "fullName": "John Doe",
               "organization": "Acme Corp."
            }
         }
      }
   ]
}
```

The same request can be sent as POST with Content-Type: application/json and this POST body:

```
{
   "query": "file",
   "pageNum": 0,
   "pageSize": 10
}
```

The query format is that used by the indexing backend, which is generally the inter-compatible Lucene/Solr/Elasticsearch format for fielded search. The fields include the payload name for pay-
loads, and modified JSON Pointers for components of the object JSON content, where array indices
are replaced with “_” as a wildcard. Typical field names are “/id”, “/name”, “/creator/organization”,
“/users/_/id”. The special field names “id”, “type”, “aclRead”, and “aclWrite” can be used to search
for objects by id, type, and acl. The special sort field name “score” can be used to sort by score as
determined by the indexing backend.

Fields under Cordra object “metadata”, “userMetadata”, and “acl” are also indexed and searchable,
using fields which are "metadata", “userMetadata”, or “acl” prepended to the JSON pointer within the
metadata or userMetadata object. Examples include "metadata/createdOn", "metadata/createdBy", "metadata/modifiedOn", "metadata/modifiedBy", "acl/readers/_", "acl/writers/_".

See Query Syntax for more details and examples for the search API.

If the boolean parameters "ids" is set, for example as GET /search?query=...&ids, then the "results" will just be a list of ids rather than a list of Cordra objects.

Note: Former versions of Cordra would return all results with pageSize=0. To restore this former behavior, you can add "useLegacySearchPageSizeZeroReturnsAll":true to the Cordra design object. By default a search with pageSize=0 returns the number of matched objects but no object content.

Request:

```
GET /search?query=file&filter=["/id","/content/name"]
```

Response:

```
{
    "size": 1,
    "pageNum": 0,
    "pageSize": -1,
    "results": [
        {
            "id": "20.5000.1234/1321d2d033b22bee1187",
            "content": {
                "name": "A file"
            }
        }
    ]
}
```

Here the filter param is used to restrict the properties in the result objects. This may be desirable if your stored objects are large and your application only requires a part of each object.

**Faceted Search**

The “facets” parameter to the search API allows faceted search, where the search results are augmented with a list of possible values that the field can take and a count of how many objects have each value.

The “facets” request parameter is a list of facet specifications, each of which has a "field" which is the index field (the JSON pointer) on which to perform faceting. A facet specification can also specify "maxBuckets", the maximum number of buckets to return; this defaults to 10.

The “facets” property of search results is a list of facet results, each of which has the "field" and a list of "buckets". Each "bucket" has a "value", a "count" and a "filterQuery". The "filterQuery" can be added verbatim to the "filterQueries" request parameter in order to restrict the search just to objects where the specified field has the specified value.

Request:
POST /search
Content-Type: application/json

{
    "pageSize": 0,
    "query": "+type:Document",
    "facets": [
    
    {
        "field": "/author"
    },
    
    {
        "field": "/status"
    }
    ]
}

Response:

{
    "size": 158,
    "facets": [
    
    {
        "field": "/author",
        "buckets": [
        
        {
            "value": "unknown",
            "count": 148,
            "filterQuery": "sort_/author:unknown"
        },
        
        {
            "value": "John Doe",
            "count": 10,
            "filterQuery": "sort_/author:"john doe"
        }
    ]
    },
    
    {
        "field": "/status",
        "buckets": [
        
        {
            "value": "release",
            "count": 80,
            "filterQuery": "sort_/status:release"
        },
        
        {
            "value": "draft",
(continues on next page)
Delete object by id

Request:

DELETE /objects/<id>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>required</td>
<td>The id of the object to delete.</td>
</tr>
<tr>
<td>jsonPointer</td>
<td>optional</td>
<td>Indicates that instead of deleting the object, the object should be modified by deleting the content at the specified JSON pointer.</td>
</tr>
</tbody>
</table>

Examples

Request:

DELETE /objects/20.5000.1234/23bdf2a62a83225a1b77

Response: empty

Delete payload by id and name

Request:

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>required</td>
<td>The id of the object containing the payload.</td>
</tr>
<tr>
<td>payload</td>
<td>required</td>
<td>The name of the payload to delete.</td>
</tr>
</tbody>
</table>
Examples

Request:

DELETE /objects/20.5000.1234/23bdf2a62a83225a1b77?payload=file

Response: empty

3.4.6.2 Handle.Net Web Proxy

Retrieve an object via the Handle.Net web proxy

Request:

GET http://hdl.handle.net/20.5000.1234/23bdf2a62a83225a1b77?locatt=view:json

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>locatt</td>
<td>optional</td>
<td>Used to specify if the redirect should respond with the json or the user interface. (default view:ui)</td>
</tr>
</tbody>
</table>

See Handle Integration for details about handle generation.

3.4.6.3 ACL API

Retrieve ACL for object

Request:

GET /acls/<id>

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>required</td>
<td>The id of the object to retrieve its acl.</td>
</tr>
</tbody>
</table>
Example

Request:

GET /acls/20.5000.1234/37b4ac94ba3e14665e04

Response:

```
{
    "readers": ["20.5000.1234/73675debcd8a436be48e"],
    "writers": ["20.5000.1234/73675debcd8a436be48e"
    ]
}
```

Update ACL for object

Request:

PUT /acls/<id>

Parameters

<table>
<thead>
<tr>
<th>id</th>
<th>required</th>
<th>The id of the object you want to set permissions on.</th>
</tr>
</thead>
</table>

Example

Request:

PUT /acls/20.5000.1234/37b4ac94ba3e14665e04

PUT Data:

```
{
    "readers": ["20.5000.1234/73675debcd8a436be48e"],
    "writers": ["20.5000.1234/73675debcd8a436be48e"
    ]
}
```
The PUT data contains two arrays, *readers* and *writers*. 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.

The standard PUT /objects update API can be used to modify ACL values as well, by including a part named "acl" in a multipart request.

**NOTE:** The ACL API has changed in Cordra 2.0. In version 1.0, the data arrays were called *read* and *write*.

Response:

```json
{
   "readers": [
      "20.5000.1234/73675debcd8a436be48e"
   ],
   "writers": [
      "20.5000.1234/73675debcd8a436be48e"
   ]
}
```

### 3.4.6.4 Password Change API

**Change password**

Request:

```
PUT /users/this/password
```

**Request Headers**

| Authorization | Should be a Basic auth header or Bearer auth header with a JWT |

Changing a password requires using the Authorization header directly authenticating the user, instead of an access token. The body of the request should just be the new password.

**Example**

Request:

```
PUT /users/this/password
```

PUT Data:

```
newPassword
```

Response:
3.4.6.5 Check Credentials API

Check Credentials

Request:

POST /check-credentials

Request Headers

Authorization Should be a Basic or Bearer auth header. Optional.

This call can be used to get information about the provided credentials. It can be used to check credentials for the given Authorization header, either for a direct authentication or for an access token. (It can also be used with the legacySessionsApi to check credentials of a cookie-based session.)

Example

Request:

GET /check-credentials

Parameters

<table>
<thead>
<tr>
<th>full</th>
<th>optional</th>
<th>If ?full=true is specified as a query parameter, additional fields are included in the response.</th>
</tr>
</thead>
</table>

Response Attribute Name | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>Whether or not the authentication was successful.</td>
</tr>
<tr>
<td>username</td>
<td>Username of the authenticated user.</td>
</tr>
<tr>
<td>userId</td>
<td>Userld of the authenticated user.</td>
</tr>
<tr>
<td>typesPermittedToCreate</td>
<td>List of types this user can create; included when $full=true is specified.</td>
</tr>
<tr>
<td>groupIds</td>
<td>List of groups this user is in; included when $full=true is specified.</td>
</tr>
</tbody>
</table>
3.4.6.6 Type Methods API

See Type Methods for details about the server-side implementation of type methods.

Call Type method

Request:

```
POST /call?objectId=<objectId>&method=<method>&attributes=<attributes>
POST /call?type=<type>&method=<method>&attributes=<attributes>
```

Some Type methods may allow the use of GET rather than POST.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>optional</td>
<td>The id of the object on which to call an instance method. Either objectId or type is required.</td>
</tr>
<tr>
<td>type</td>
<td>optional</td>
<td>The type on which to call a static method.</td>
</tr>
<tr>
<td>method</td>
<td>required</td>
<td>The name of the method to call.</td>
</tr>
<tr>
<td>attributes</td>
<td>optional</td>
<td>A JSON object, possibly including a filename, a media type, and a request context. The attributes and filename, media type, and request context are made available to schema methods.</td>
</tr>
</tbody>
</table>

The POST body is parsed as JSON and passed to the method as context.params. Alternately the method can use context.directIo to access potentially non-JSON input (see Direct I/O).

If the method is accessed via GET (or POST with empty body) the input can be given by the URI query parameter "params".

Call requests allow a query parameter called attributes. This must be a valid JSON object (suitably encoded as a query parameter). For convenience, attributes as a JSON object can be expressed as multiple properties in the API request with dot separated names. If the value of some attributes.prop is valid JSON, then it is parsed as JSON. Otherwise, the value is interpreted as a string. For example,

```
GET /call?...&attributes={"query":"foo","pageSize":10}
```

is exactly the same as

```
GET /call?...&attributes.query=foo&attributes.pageSize=10
```

By default this API requires write permission on the object or schema. ACLs for calling methods can be configured as described in Authorization for Type Methods.
List Type methods

Request:

GET /listMethods/?objectId=<objectId>
GET /listMethods/?type=<type>
GET /listMethods/?type=<type>&static

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>optional</td>
<td>The id of the object you want to list methods of. Either objectId or type is required.</td>
</tr>
<tr>
<td>type</td>
<td>optional</td>
<td>A Cordra type; depending on the static parameter, this will list static methods on that type, or instance methods on objects of the type.</td>
</tr>
<tr>
<td>static</td>
<td>optional</td>
<td>If present, listing methods for a type will list static methods instead of instance methods.</td>
</tr>
</tbody>
</table>

The HTTP response is a list of strings which are the available method names.

Batch Upload

Request:

POST /batchUpload
POST /batchUpload?format=ndjson
POST /batchUpload?failFast=true
POST /batchUpload?parallel=false

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>optional</td>
<td>Set to ndjson to indicate that the format of the request body is newline-delimited JSON objects.</td>
</tr>
<tr>
<td>failFast</td>
<td>optional</td>
<td>If ?failFast=true is specified as a query parameter, the server will stop processing the input on the first encountered error.</td>
</tr>
<tr>
<td>parallel</td>
<td>optional</td>
<td>If ?parallel=false is specified as a query parameter, the server will process each object supplied in the input on a single thread in order.</td>
</tr>
</tbody>
</table>
Example

Request:

POST /batchUpload?format=ndjson

POST Data:

{"type":"Foo","content":{}}
{"id":"test/e4673f57012b544e72af", "type":"Foo","content":{}}

Response:

{  "success": true
}

The input may optionally include an id on each object, as shown in the first example. For each object in the request Cordra will first try to create that object. If the create operation fails because an object with that id already exists an update will be performed.

By default the input objects will be processed in parallel in a multi-threaded fashion. As such they may not be processed in the order they are supplied in the input. If you need the input objects processed in order you can specify the optional query param parallel=false. It would then be the responsibility of the caller to ensure any dependencies are correctly ordered. For example if you are creating an object of type Foo and the Schema Foo does not yet exist, include the Schema object in the request before any instances that use it.

An alternative approach is to send multiple batch requests to the server. For example a first batch might include all the Schema objects and once complete, a second batch could contain all the instances of those types. Here each individual batch could safely be processed in parallel.

There are three supported request input formats:

Newline-Delimited JSON:

{...
{...
{...}

Cordra Search Results:

{  "results": [
   {
      ...
   },
   {
      ...
   },
   ...
  ]

(continues on next page)
Cordra Object JSON Array:

```json
[
  {
    ...
  },
  {
    ...
  },
  {
    ...
  }
]
```

By default the API expects either Cordra Object JSON Array or Cordra Search Results. You do not need to specify which as Cordra will detect between those two formats automatically. If you supply Newline-Delimited JSON you must include the `?format=ndjson` query param with the request.

Payload data may be included directly in the JSON as base64 encoded strings using the property name `base64Payload`. For example:

```json
[
  {
    "id": "test/123",
    "type": "Foo",
    "content": {},
    "payloads": [
      {
        "base64Payload": "TG9yZW0gaXBzdW0gZG9sb3Igc2l0IGFtZXQsIGNvbnNlY3RldHVyIGFkaXBpc2NpbmcgZWxpdA==",
        "name": "payload",
        "filename": "Loremipsum.txt",
        "mediaType": "text/plain",
        "size": 55
      }
    ]
  }
]
```

It is not necessary to include the `base64Payload` when updating an object. If `base64Payload` is missing on an update the stored payload will remain unchanged. However if you want to delete a pay-
load during an update you can specify the payload names to be deleted in a top level property payloadsToDelete as shown below:

```json
[
  {
    "id": "test/123",
    "type": "Foo",
    "content": {},
    "payloadsToDelete": [
      "loremipsum"
    ]
  }
]
```

### 3.4.6.7 Access Token API

**Warning:** The following API was introduced in Cordra v2.0.0. If you are using an earlier version, please refer to legacySessionsApi for information on using cookie-based sessions with the HTTP REST API.

The access token API can be used to authenticate only once to obtain an access token, which can then be provided for multiple calls. This way, the server does not need to reprocess your authentication credentials with each request, which can significantly speed up the process of making multiple requests.

Once authenticated using the access token API, the access token should be sent to other APIs using an Authorization Bearer header. For example:

```text
Authorization: Bearer ACCESS_TOKEN
```

#### Create a new access token

**Request:**

```
POST /auth/token
```

**POST Data:**

```json
{
  "grant_type": "password",
  "username": <username_or_userid>,
  "password": <password>,
}
```

or
The POST data specifies whether the user is authenticating via username/password or via keypair, depending on the grant_type. If the grant_type is password, the POST data should contain the username (or user object id) and password. See Authentication via Keys for the details of the JWT assertion that must be included with keypair authentication.

Note that with some tools you must specify a header ‘Content-Type: application/json’ in order to have the POST body correctly interpreted as JSON.

Password authentication allows specifying either a username, or the Cordra object id of the user object. (If the two possible interpretations indicate two different Cordra objects, the Cordra object id will be used.)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>optional</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Response Attribute Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access_token</td>
<td>The newly created access token.</td>
</tr>
<tr>
<td>token_type</td>
<td>Always “Bearer”.</td>
</tr>
<tr>
<td>active</td>
<td>Whether or not the token is active; always “true” for successful calls of the /auth/token API.</td>
</tr>
<tr>
<td>username</td>
<td>Username of the authenticated user</td>
</tr>
<tr>
<td>userId</td>
<td>UserId of the authenticated user</td>
</tr>
<tr>
<td>typesPermittedToCreate</td>
<td>List of types this user can create; included when <code>?full=true</code> is specified.</td>
</tr>
<tr>
<td>groupIds</td>
<td>List of groups this user is in; included when <code>?full=true</code> is specified.</td>
</tr>
</tbody>
</table>

**Get access token information**

**Request:**

```json
POST /auth/introspect
{
  "token": <token>
}
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>optional</td>
<td>If ?full=true is specified as a query parameter, additional fields are included in the response.</td>
</tr>
</tbody>
</table>

Response Attribute Name | Description
---|---
active | Whether or not the token is active.
username | Username of the authenticated user
userId | UserId of the authenticated user
typesPermittedToDoCreate | List of types this user can create; included when ?full=true is specified.
groupIds | List of groups this user is in; included when ?full=true is specified.

Delete specified access token

Request Headers

Authorization | Bearer <token>

Request:

POST /auth/revoke

{
    "token": <token>
}

Response:

{
    "active": false
}
3.4.7 Administrative APIs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /schemas/&lt;type&gt;</td>
<td>Get schema or all schemas.</td>
</tr>
<tr>
<td>PUT /schemas/&lt;type&gt;</td>
<td>Create or update a schema.</td>
</tr>
<tr>
<td>DELETE /schemas/&lt;type&gt;</td>
<td>Delete a schema.</td>
</tr>
<tr>
<td>POST /uploadObjects</td>
<td>Bulk upload of objects from a json file.</td>
</tr>
<tr>
<td>PUT /adminPassword</td>
<td>Change the password for the admin user.</td>
</tr>
<tr>
<td>POST /updateHandles</td>
<td>Update all handle records for objects in Cordra.</td>
</tr>
<tr>
<td>GET /updateHandles</td>
<td>Get status of handle update.</td>
</tr>
<tr>
<td>POST /reindexBatch</td>
<td>Reindex all specified objects.</td>
</tr>
<tr>
<td>GET /startupStatus</td>
<td>Accessible even after failed startup to indicate success or failure.</td>
</tr>
</tbody>
</table>

3.4.7.1 Schemas API

Retrieve schema

Request:

GET /schemas
GET /schemas/<type>

Example

Request:

GET /schemas/JavaScriptDirectory

Response:

```json
{
    "type": "object",
    "required": [
        "directory"
    ],
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "type": 
```
Create or update schema

Request:

```
PUT /schemas/<type>
```

Example

Request:

```
PUT /schemas/JavaScriptDirectory
```

Request Body:

```json
{
    "type": "object",
    "required": [
        "directory"
    ],
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "type": {
                    "autoGeneratedField": "handle"
                }
            }
        }
    }
}
```

(continues on next page)
Response:

```
{
  "msg": "success"
}
```

Delete schema

Request:

```
DELETE /schemas/<type>
```

Example

Request:

```
DELETE /schemas/JavaScriptDirectory
```

Response:

```
{
  "msg": "success"
}
```
3.4.7.2 Upload Objects API

Upload objects

Request:

POST /uploadObjects

Parameters

| deleteCurrentObjects | boolean | If true, delete all existing objects before uploading new objects. Otherwise, update objects. Default: false |

Example

Request:

POST /uploadObjects

Request Body:

```json
{
    "results": [
        {
            "id": "test/171a0606f7c74580fd39",
            "type": "Schema",
            "content": {
                "identifier": "test/171a0606f7c74580fd39",
                "name": "Group",
                "schema": < Schema json omitted for brevity >,
                "javascript": < JavaScript omitted for brevity >
            },
            "metadata": {
                "createdOn": 1535479938849,
                "createdBy": "admin",
                "modifiedOn": 1535479938855,
                "modifiedBy": "admin",
                "txnId": 65
            }
        },
        {
            "id": "test/171a0606f7c74580fd39",
            "type": "Schema",
            "content": {
                "identifier": "test/171a0606f7c74580fd39",
                "name": "Group",
                "schema": < Schema json omitted for brevity >,
                "javascript": < JavaScript omitted for brevity >
            },
            "metadata": {
                "createdOn": 1535479938849,
                "createdBy": "admin",
                "modifiedOn": 1535479938855,
                "modifiedBy": "admin",
                "txnId": 65
            }
        }
    ]
}
```

(continues on next page)
3.4.7.3 Admin Password API

Used to update the admin user password.

**Update admin password**

**Request:**

```
PUT /adminPassword
```

**Example**

**Request Body:**

```json
{
   "password": "newPassword"
}
```

**Response:**

```json
{
   "success": "true"
}
```
3.4.7.4 Update Handles API

Used to update the admin user password.

Start handle update

Request:

POST /updateHandles

Example

Response:

{}  

Get status of handle update

Request:

GET /updateHandles

Example

Response:

{
   "inProgress": true,
   "total": 123,
   "progress": 52,
   "startTime": 1535479938855,
   "exceptionCount": 0
}

3.4.7.5 Reindex Batch API

Used to reindex the specified list of objects. Takes a JSON array of object ids to be reindexed. When using object locking, which is on by default, batch sizes should be small e.g. 16. However many reindex requests can be sent in parallel.

An optional “query” query param can be sent instead of a list of ids in the body. The query will be executed and the results will be reindexed. The results will be reindexed by a single thread in batches of size 16.
Reindex batch of objects

Request:

POST /reindexBatch

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock-objects</td>
<td>Defaults to ‘true’. Locks on object ids while objects are reindexed. You should only set this to ‘false’ if users are not using the system during reindexing, or it is otherwise possible to guarantee that the objects being reindexed will not be concurrently updated. If this is possible, performance is improved by setting this to false.</td>
</tr>
<tr>
<td>query</td>
<td>A lucene query that can be sent instead of a list of ids in the body.</td>
</tr>
<tr>
<td>all</td>
<td>If set, all objects will be reindexed from storage.</td>
</tr>
</tbody>
</table>

Example

Request Body:

```
[
    "test/abc",
    "test/def",
    "test/xyz"
]
```

Response:

```
{
    "success": "true"
}
```

3.4.7.6 Startup Status API

This API provides some information about how far startup progressed, even in the event of startup failure. This is useful in installations where remote access is easier than directly looking at log files; however, the amount of information provided is very limited. It will contain a "state" which is either "UP" or "FAILED", and a "details" which will contain the status ("UP" or "FAILED") of some of the following, depending on your Cordra configuration:

- "storage": Cordra’s configured storage module
- "indexer": Cordra’s configured indexer module
- "zookeeper": subsystem for accessing ZooKeeper (for a distributed Cordra)
• “replicationProducer”: subsystem for sending transactions to be replicated (using Kafka)
• “replicationConsumer”: subsystem for receiving transactions to be replicated (using Kafka)

If the “state” is “FAILED”, this request will return an HTTP 500 response for ease of monitoring.

Request:

GET /startupStatus

Response:

```json
{
  "state": "UP",
  "details": {
    "storage": "UP",
    "indexer": "UP"
  }
}
```

### 3.4.8 Experimental APIs

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET /versions/?objectId=&lt;objectId&gt;</td>
<td>Get version information for a specific object.</td>
</tr>
<tr>
<td>POST /versions/?objectId=&lt;objectId&gt;</td>
<td>Create a version of a specific object.</td>
</tr>
</tbody>
</table>

#### 3.4.8.1 Versioning API

**Retrieve object version**

Request:

GET /versions/?objectId=<objectId>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>required</td>
</tr>
</tbody>
</table>
Example

Request:

GET /versions/?objectId=20.5000.1234/eb3b797f9fd544fb90fb

Response:

```
[
    {
        "id": "20.5000.1234/208b07aec73a36b91a1b",
        "type": "Foo",
        "versionOf": "20.5000.1234/eb3b797f9fd544fb90fb",
        "publishedBy": "admin",
        "publishedOn": 1436380157539,
        "isTip": false
    },
    {
        "id": "20.5000.1234/eb3b797f9fd544fb90fb",
        "type": "Foo",
        "modifiedOn": 1433957772377,
        "isTip": true
    }
]
```

Create object version

Request:

POST /versions/?objectId=<objectId>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>required The id of the object you want to create a version of.</td>
</tr>
<tr>
<td>versionId</td>
<td>optional The desired id of the new version. If omitted Cordra will mint an id.</td>
</tr>
<tr>
<td>clonePayloads</td>
<td>optional If present, the new version object will not contain a copy of the payloads.</td>
</tr>
</tbody>
</table>
Example

Request:

POST /versions/?objectId=20.5000.1234/eb3b797f9fd544fb90fb

Response:

```json
{
   "id": "20.5000.1234/37b4ac94ba3e14665e04",
   "type": "Foo",
   "versionOf": "20.5000.1234/eb3b797f9fd544fb90fb",
   "publishedBy": "admin",
   "publishedOn": 1436380685442,
   "isTip": false
}
```

3.5 Query Syntax

Cordra's APIs enable users to issue queries to search across the managed digital objects based on certain criteria.

3.5.1 queryJson

As of Cordra v2.1.0 search requests can be sent with a "queryJson" parameter. The value of the parameter should be a JSON object. Matching objects are those whose content matches all the JSON provided in the "queryJson" parameter.

Example:

```json
{
   "queryJson": {
      "name": "foo"
   }
}
```

This will match any Cordra object the content of which has a top-level property "name", the value of which contains the token "foo". Deeper structure within queryJson can be used to match properties below the top level.
3.5.2 query

The remainder of this section discusses instead the more general “query” parameter. We describe via examples the query syntax to follow to retrieve desired results from queries.

Cordra uses the Lucene query syntax for search. Details of that syntax can be found at Lucene Query Syntax.

The examples below demonstrate the query syntax for fields in the following Cordra object that represents metadata about the book “Tess of the D’Urbervilles”.

```
{
    "id": "test/72d1c8508991c7aa0a22362de8574f9c4a0fd28e7ac5bfb4002522b1b7aabafa",
    "type": "Book",
    "content": {
        "title": "Tess of the D’Urbervilles",
        "description": "Tess Durbeyfield is driven by family poverty to claim kinship with the wealthy D’Urbervilles and seek a portion of their family fortune.",
        "author": {
            "firstName": "Thomas",
            "lastName": "Hardy"
        },
        "genre": [
            "Victorian",
            "Tragedy"
        ],
        "publishers": [
            {
                "by": "James R. Osgood, McIlvaine & Co.",
                "date": "1891"
            },
            {
                "by": "Penguin Classics",
                "date": "2003"
            }
        ],
        "language": "English",
        "displayDateTime": "2022-08-08T09:00:00.000-04:00"
    },
    "acl": {
        "writers": [
            "test/xyz"
        ]
    },
    "userMetadata": {
        "Foo": "Bar"
    },
    "metadata": {
        (continues on next page)
    }
}
```
The simplest type of query is one that contains one or more terms that are located anywhere in the object.

For example the above object would be included in results for the query:

```
Tess
```

Multiple terms can be combined in a space separated list:

```
Tess Durbeyfield Tragedy
```

These terms are combined with a logical OR such that the above query would match any object that contained any of these three terms.

### 3.5.2.1 Phrase Query

Terms can be grouped together into phrases using double quotes. Here the query:

```
"family poverty"
```

would match to our example object but the query:

```
"poverty family"
```

would not, because the terms in the object are not in the same order as they are in the phrase query.

Note that terms will be tokenized on some non-whitespace characters. For example the query `foo-bar` would match all of foo, bar, and bar-foo as well. If you only wanted to match `foo-bar`, you must explicitly wrap the query in quotes to make it a phrase query, e.g. `"foo-bar"`. 

---

Query Syntax
3.5.2.2 Fielded Queries

Fields allow the search to be restricted to particular properties of digital objects. The JSON schema driven portion of the object called "content" is fielded using JsonPointers. These are slash separated paths into the JSON tree followed by a colon and then the term. The following example JsonPointer terms would match the example object:

```
/description:family
/title:Tess
date_metadata/createdOn:2022-08-08
```

The terms for the properties of sub-JSON-objects are defined with slashes:

```
/author/lastName:Hardy
```

Fielded queries can also be combined with boolean operators:

```
/author/lastName:Hardy AND /author/firstName:Thomas
```

Such a query would only match an object that had both a lastName Hardy and a firstName Thomas.

3.5.2.3 Wild card queries

A * character can be used to find results where only part of the term matches:

```
/author/lastName:Har*
/author/lastName:H*y
```

/author/lastName:*y is an invalid query in a default single-instance Cordra. It is supported when Cordra is using a Solr or Elasticsearch backend, or when the Lucene backend has been configured to support it; see Configuring Indexing Backend.

3.5.2.4 Fuzzy matching

Fuzzy matching allows for small corrections in spelling mistakes. Here, the below incorrect spelling of Hardy will still match the example object:

```
/author/lastName:Hardi~
/author/lastName:Hardie~
```

Fuzzy queries only match terms that are different from the query by at most two characters.
3.5.2.5 Searching arrays

In order to explicitly search for the term “Tragedy” within the array property named “genre” the underscore character is used:

```
/genre/_:Tragedy
```

In order to search for properties on objects which themselves are in an array, such as the publishers array, e.g. search for all books with a publisher by “Penguin”:

```
/publishers/_/by:Penguin
```

3.5.2.6 Range Queries

To search for objects that have a value that falls between two values is called a range query. The below example shows a range query on the date field. It will match any value between 2000 and 2004 inclusively:

```
date_/publishers/_/date:[2000-01-01 TO 2004-12-31]
```

The same query but excluding the upper and lower bounds uses curly brackets:

```
date_/publishers/_/date:{2000-01-01 TO 2004-12-31}
```

Wild cards can also be used to search for anything less than:

```
date_/publishers/_/date:[* TO 2004-12-31]
```

Or anything greater than:

```
date_/publishers/_/date:[2000-01-01 TO *]
```

Note that for most fields range values are treated as text such that less than and greater than refer to lexicographical ordering. When a `num_` or `date_` prefix is included in a query key, range values are treated as numeric or as date-times respectively.

3.5.2.7 Querying numeric properties

By default all properties of the content of an object are indexed as text. This has implications on the order of those values especially when considering numbers in range queries. Range queries over text fields use the lexicographical (dictionary) order of the text. For example the following numbers as text “1”, “10”, “2” are in lexicographical order. Consider the follow object:

```
{
    "foo": 2
}
```

And the range query:
The above query would not include our object in the results because in lexicographical order 2 comes after 10. It is often useful to have numeric values sorted in numerical order 1, 2, 10 instead. To support this Cordra creates an additional field for every JSON number it finds in the content when indexing. This additional field is prepended with the prefix num_. As such you can use the following range query which will include our object:

```
num_/foo:[1 TO 10]
```

This additional field is indexed as double precision floating point number. Note that you do not need to indicate anything in the schema to get numeric fields indexed this way. All numbers in the JSON get this extra field automatically.

### 3.5.2.8 Querying date-time properties

When the value of a JSON property is a string with the ISO 8601 date-time format:

```
2022-01-05T08:30:00.000-05:00
```

then an additional field is created, prepended with the prefix date_. The date_ fields can be used to ensure the punctuation does not lead to tokenization, and for range querying; for example:

```
date_/foo:[2022-01-01 TO 2022-01-05]
```

will match the preceding value.

date_ fields will be created for fields which only have a year-month-day, or only year-month-day and hour:minute; fractional seconds can be omitted or included; the time zone can be omitted or without colons or can be “Z”; instead of the ISO-8601-specified “T” a space can be used to separate the date and the time. (Range queries need to use the “T”.) Additionally, the fields date_metadata/createdOn, date_metadata/modifiedOn, and data_metadata/publishedOn are created.

### 3.5.2.9 Query if a property exists

This can be achieved by performing a wildcard range query from any value to any value. The query below will return all objects that have a property /language regardless of the value:

```
/language:[* TO *]
```
3.5.2.10 Metadata

Cordra managed metadata of a digital object is also managed as JSON. That metadata is a sibling of content within the Cordra object. Properties within metadata can be searched by property name by prefixing it with "metadata":

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata/createdOn</td>
<td>1562866891119</td>
</tr>
<tr>
<td>metadata/modifiedOn</td>
<td>1562945123652</td>
</tr>
<tr>
<td>metadata/createdBy</td>
<td>admin</td>
</tr>
<tr>
<td>metadata/modifiedBy</td>
<td>admin</td>
</tr>
<tr>
<td>metadata/txnId</td>
<td>1562945123643011</td>
</tr>
</tbody>
</table>

If hashes have been turned on for this type of object those can also be searched on:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata/hashes/full</td>
<td>58848eeda8472a14f4c5fb709aa96094409018b0e623baf7c94c991ea3811f15</td>
</tr>
</tbody>
</table>

Some parts of the metadata can be searched with special field names:

Search by type:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Book</td>
</tr>
</tbody>
</table>

Search by id:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>&quot;123/test&quot;</td>
</tr>
</tbody>
</table>

Search by the user that created or modified the object:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createdBy</td>
<td>admin</td>
</tr>
<tr>
<td>modifiedBy</td>
<td>admin</td>
</tr>
</tbody>
</table>

Creation and modification timestamps:

The two fields objcreated and objmodified contain the timestamp of the object converted into human readable format yyyyMMddHHmmssSSS. Note that this field does not contain delimiters. Delimiters can result in tokenization of the string which can then be challenging to search on:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>objcreated</td>
<td>yyyyMMddHHmmssSSS</td>
</tr>
<tr>
<td>objmodified</td>
<td>yyyyMMddHHmmssSSS</td>
</tr>
</tbody>
</table>

If the object contains userMetadata, as it does in this case, it can be searched with the "userMetadata" prefix:
If explicit ACLs have been added to the object those can also be searched as JSON with the "acl" prefix. For example searching for all objects that have given explicit write permission to test/xyz:

```
 acl/writers/_:"test/xyz"
```
4.1 Introduction

The Cordra software distribution includes three client libraries. Software developers building applications using Java may use either the DOIP Client Library - Java Version or the Cordra HTTP REST Client Library - Java Version. JavaScript and TypeScript software developers who are interested in building web applications may use the JavaScript Version.

4.2 Cordra HTTP REST Client Library - Java Version

A Cordra HTTP REST client library in Java for interacting with a Cordra service is included in the distribution. This library is specifically designed for use with the Cordra's REST API for interacting with Cordra.

4.2.1 Location

The client library can be included in your Gradle or Maven build files.

Gradle:

```
compile group: 'net.cnri.cordra', name: 'cordra-client', version: '2.5.0'
```

Maven:

```
<dependency>
  <groupId>net.cnri.cordra</groupId>
  <artifactId>cordra-client</artifactId>
  <version>2.5.0</version>
</dependency>
```

The client library and its dependencies can also be found in this directory in the downloaded package:

```
corda/sw/lib/cordra-client/
```
4.2.2 Background

The various methods supported by the Cordra Java client library are defined in this Java Interface:

```java
net.cnri.cordra.api.CordraClient
```

There are two implementations of that Java Interface, one that does not use access tokens and one that does:

```java
net.cnri.cordra.api.HttpCordraClient
net.cnri.cordra.api.TokenUsingHttpCordraClient
```

The basic approach to using Cordra would be to leverage any of the above implementations to perform create, retrieve, update, and delete (CRUD) operations and other operations. See the code in the aforementioned Java classes for details on the various supported operations.

That said, the basic digital object Java class stated below should be used for adding, removing, or updating information prior to invoking any Cordra client operations to commit the changes on the Cordra service:

```java
net.cnri.cordra.api.CordraObject
```

4.2.3 Example Usages

- Create a new instance of CordraClient:

```java
String baseUri = "https://localhost:8443/";
String username = "admin";
String password = "password";

CordraClient cordra = new TokenUsingHttpCordraClient(baseUri, username, password);
```

- Create a digital object of type Document (without attaching any payload, for brevity sake):

```java
JsonObject doc = new JsonObject();
doc.addProperty("name", "example name");
doc.addProperty("description", "description");
CordraObject co = new CordraObject();
co.id = "test/123";
co.setContent(doc);
co.type = "Document";

co = cordra.create(co);
```

- Retrieve the digital object:
CordraObject co = cordra.get("test/123");

• Update the object:

CordraObject co = cordra.get("test/123");
co.content.getAsJsonObject().addProperty("name", "updated example name");
co = cordra.update(co);

• Delete the object:

cordra.delete("test/123");

• Search for objects:

try (SearchResults<CordraObject> results = cordra.search("*:*")) {
    for (CordraObject co : results) {
        System.out.println(co.id);
    }
}

• Invoke a Type method (instance method):

JsonElement result = cordra.call("test/123", "exampleInstanceMethod", params, options);
CallResponse response = cordra.callAsResponse("test/123", "exampleInstanceMethod", input, options);

• Invoke a Type method (static method):

JsonElement result = cordra.callForType("Member", "getMemberFromUsername", params, options);
CallResponse response = cordra.callForTypeAsResponse("Member", "getMemberFromUsername", input, options);

The third argument of the various call methods can either be a JsonElement or an InputStream. If the AsResponse form is used, the return type allows access to response.headers.filename and response.headers.mediaType (if set by the type method) and well as response.body as an InputStream. The CallResponse object is Closeable.
4.2.4 TLS Configuration

By default, instances of HttpCordraClient will connect to any server over HTTPS without authenticating the server. This behavior can be controlled by system properties as follows.

<table>
<thead>
<tr>
<th>System Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cor-dra.client.tls.hostnameVerification</td>
<td>Must be set to true in order for CordraClient to verify the server hostname against the server certificate; the default is no hostname verification.</td>
</tr>
<tr>
<td>cor-dra.client.tls.useSystemDefault</td>
<td>Use the Java default behavior for trusting server certificates, including referring to javax.net.ssl.* system properties.</td>
</tr>
<tr>
<td>cor-dra.client.tls.useDefault</td>
<td>Use the Java default behavior for trusting server certificates, but ignoring the javax.net.ssl.* system properties.</td>
</tr>
<tr>
<td>cor-dra.client.tls.trustStore</td>
<td>Filesystem path to a keystore which defines which Cordra HTTPS certificates to trust.</td>
</tr>
<tr>
<td>cor-dra.client.tls.trustStorePassword</td>
<td>Password for the trustStore.</td>
</tr>
</tbody>
</table>

4.3 Cordra Client Library - JavaScript Version

This Cordra Client Library can be used to develop JavaScript and TypeScript applications that are based on Cordra.

This library supports browsers and Node.js. All modern browsers are supported including IE11 (that leverages the included fetch polyfill).

The client library is written in TypeScript; type definitions are included in the release distribution for use in your applications.

4.3.1 Installation

You can install the client library using npm as follows:

```
npm install @cnri/cordra-client
```

The following artifacts are included in the distribution:

- **dist/esm/**, **dist/cjs/**: Node.js version of cordra-client library, with ESM and CJS support.
- **dist/cordra-client.js**, **dist/cordra-client.min.js**: Browser versions of the library suitable for including in `<script>` HTML tags, both un-minimized and minimized. These can also be used as a CommonJS module. Copies are supplied with the version number in the filename.
- **dist/cordra-client.esm.js**: Browser version of the library as a standard ES module, suitable for use in applications built using an ESM-aware module bundler like Rollup or webpack 2+.
- **dist/types/**: A directory of `.d.ts` files for use with TypeScript projects.

The `.js` files all come with corresponding `.js.map` JavaScript source map files.
4.3.2 Getting Started

This Cordra client library is simple to get started with. Here is an example of searching Cordra for all Schema objects:

```javascript
import { CordraClient } from '@cnri/cordra-client';

const client = new CordraClient("https://localhost:8443");
client.search("type:Schema")
 .then(response => {
    console.log("Number of results: " + response.size);
    response.results.forEach(result => { console.log(result.content.name) });
  });
```

This will print the name of each Schema object to the console.

4.3.3 Authentication

Cordra client methods generally take a final optional Options argument which defines the user credentials used for sending the request to Cordra. The default options can be set when the CordraClient is instantiated, or later by setting the property defaultOptions.

If no defaultOptions is set, calls will be made anonymously.

Cordra client tracks authentication tokens automatically. If you wish to check whether an authentication token can be successfully obtained, call client.authenticate(options). Otherwise authentication tokens will be automatically obtained as needed.

Example Password Authentication

```javascript
const options = {
    username: "testUser",
    password: "password"
};
const client = new CordraClient(cordraBaseUri, options);
```

Example Private Key Authentication

```javascript
const privateKey = {
    "kty": "RSA",
    "n": "4zExVqSPDNAIooQyNDm_˓→g8ew9RwdDcRCuWBjIZrfIHVGLJn1VbT4reseduDJ0MVELdDp64RTH8jVxboWQIpQ",
    "e": "AQAB",
    "d": "CPmfhkMzhbdMmFC1-wjtpym3wGq7CoxGWvvNEGOV2h47gJaMBAsh4XYszToaNOK0g-˓→OpCQ73dn8FsvIKmh5VQQ",
    "p": "-_sgNIghoOHPnYjmcsQ9VXL73oGQqtVd48C8ZjmFE",
    "q": "5EEdc10pHn1-p79KtefVPwVFMUJT0QKG-BfJfxXaXU",
    "dp": "VAbKPgUjyiykWALEKKhDKCFBCfnmgEbtwapY95yqme",
};
```

(continues on next page)
4.3.4 API Docs

Full API documentation is available here.

4.4 DOIP Client Library - Java Version

The Cordra software includes the reference implementation software for the DOIP v2 specification, as provided by the DONA Foundation. Much of this DOIP documentation is based on the documentation provided by the DONA Foundation.

The DOIP client software can be used to interact with Cordra's DOIP Interface.

4.4.1 Location

The DOIP client library can be included in your Gradle or Maven build files.

Gradle:

```
compile group: 'net.dona.doip', name: 'doip-sdk', version: '2.1.0'
```

Maven:

```
<dependency>
  <groupId>net.dona.doip</groupId>
  <artifactId>doip-sdk</artifactId>
  <version>2.1.0</version>
</dependency>
```

The client library and its dependencies can also be found in the downloaded package in the following directories:

```
cordra/sw/lib/
cordra/sw/lib/cordra-client/
```
4.4.2 Background

The DOIP client provides both TransportDoipClient, which includes Java classes that closely reflect the structures of the DOIP v2.0 specification, and DoipClient, which uses TransportDoipClient to expose a Java API that maps more directly with the conceptual usage of the DOIP with one Java method per basic operation, and one more for invoking any extended operation.

We only discuss the DoipClient here. Please build Javadoc for the Cordra software and refer to it for details on TransportDoipClient.

4.4.3 DoipClient

This client serves two purposes. First, it provides a Java API to invoke the basic operations as well as any extended operations, abstracting away the protocol-specific serialization details. And secondly, it uses an Identifier/Resolution mechanism, in particular the Handle System, to discover where to send DOIP requests at request time using the target Id of the digital object being interacted with. Additionally, the client maintains connection pools for the DOIP services being interacted with.

Constructing an instance of the DoipClient is done with a no-args constructor. There is no need to specify the IP address or port of the DOIP server to be communicated with as that information is discovered automatically when the request is made. However, should there be a need to explicitly instruct the client to communicate with a specific DOIP service, ServiceInfo can be optionally supplied with each request.

An example to create a digital object is shown below:

```java
DoipClient client = new DoipClient();
AuthenticationInfo authInfo = new PasswordAuthenticationInfo("admin", "password");

DigitalObject dobj = new DigitalObject();
dobj.id = "35.TEST/ABC";
dobj.type = "Document";
JsonObject content = new JsonObject();
content.addProperty("name", "example");
dobj.setAttribute("content", content);

Element el = new Element();
el.id = "file";
el.in = Files.newInputStream(Paths.get("/test.pdf");
dobj.elements = new ArrayList<>();
dobj.elements.add(el);

ServiceInfo serviceInfo = new ServiceInfo("35.TEST/DOIPServe");

DigitalObject result = client.create(dobj, authInfo, serviceInfo);
```
4.4.3.1 ServiceInfo

The ServiceInfo contains information specifying where to send the request. This can be a handle that resolves to a handle record that contains a value for type 0.DOIP/ServiceInfo, the structure of which is defined in the DOIP specification. Alternatively, the ServiceInfo could directly contain the IP address, port and service id of the target service. In the above example a handle is supplied to direct the client to the service that the DigitalObject should be created at.

A call to retrieve a DigitalObject with identifier that resolves to the service information would look like this:

```java
DigitalObject result = client.retrieve("35.TEST/ABC", authInfo);
```

Here, only the Id of the object is passed into the call and the ServiceInfo is discovered by the client. Other operations that apply to existing objects could be invoked without supplying ServiceInfo.

However, ServiceInfo can be supplied specifically as shown in the below example:

```java
ServiceInfo serviceInfo = new ServiceInfo("35.TEST/DOIPServer", "10.0.1.1", 8888);
DigitalObject result = client.retrieve("35.TEST/ABC", authInfo, serviceInfo);
```

4.4.3.2 AuthenticationInfo

Three classes are provided that can be used to send authentication information to the DOIP server.

- PasswordAuthenticationInfo, which sends a username and password.
- PrivateKeyAuthenticationInfo, which given a Privatekey, will generate and send a JSON Web Token (RFC 7519).
- TokenAuthenticationInfo, which given a (any) token, will send it with the request.

4.4.3.3 Basic Operations

Methods are provided that support the 7 basic operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.DOIP/Op.Hello</td>
<td>hello</td>
</tr>
<tr>
<td>0.DOIP/Op.Create</td>
<td>create</td>
</tr>
<tr>
<td>0.DOIP/Op.Retrieve</td>
<td>retrieve, retrieveElement</td>
</tr>
<tr>
<td>0.DOIP/Op.Update</td>
<td>update</td>
</tr>
<tr>
<td>0.DOIP/Op.Delete</td>
<td>delete</td>
</tr>
<tr>
<td>0.DOIP/Op.search</td>
<td>search, searchIds</td>
</tr>
<tr>
<td>0.DOIP/Op.ListOperations</td>
<td>listOperations</td>
</tr>
</tbody>
</table>
4.4.3.4 **DigitalObject**

DigitalObject is a Java class that represents the structure of a Digital Object as defined in the DOIP v2 specification. The DigitalObject Java class contains the id, type, attributes and elements. The create, update, retrieve and search methods make use of DigitalObject in their arguments and return types.

4.4.3.5 **Search**

Two search methods are provided, one that returns DigitalObject instances, and one that only returns the Ids of the objects that match the query: search() and searchIds() respectively. Both return an object of type SearchResults which implements Iterable.

```
QueryParams queryParams = new QueryParams(0, 10);
String query = "type:Document";
try (SearchResults<DigitalObject> results = client.search("35.TEST/DOIPServer", query, queryParams, authInfo)) {
    for (DigitalObject result : results) {
        System.out.println(result.id + ": " + result.type);
    }
}
```

The number of results returned are limited by what was specified in the request or what the server deems appropriate. QueryParams is used in this example to specify the server to return 10 results from offset 0. SearchResults is Autocloseable, and so can be used in a try-with-resources statement. Without a try-with-resources statement, SearchResults must be explicitly closed at the end of results processing in order to release the connection to the server back to the pool of connections managed by this client library. SearchResults is also Iterable, and so can be used in a for-in loop.

4.4.3.6 **performOperation**

All of the basic operations use the method performOperation() internally. This method is made public (in a Java sense) such that you can send extended operations.

The performOperation method takes an InDoipMessage and returns a DoipClientResponse. These classes and their use are described in the Javadoc.

4.5 **Third Party Clients**

A Cordra REST API client library in Python is being developed (in part) by a few employees at NIST. A pre-alpha version is available [here](#).
5.1 Introduction

Cordra is a highly configurable software system. You can learn about design configuration [here](#).

In general, Cordra is configured by (1) the config.json file in the data directory, or its equivalent Zookeeper node for distributed deployment; and (2) the design object. The config.json file is used to specify which storage and indexing backends Cordra will use, along with various other features which require configuration before Cordra can be up and running. When Cordra is deployed as a single instance using its own built-in servlet container, the config.json file is also used to configure the HTTP and HTTPS interfaces. Other features, which can be modified at runtime and can be determined safely after Cordra startup, are configured in the design object.

For convenience a repolInit.json file can be used to specify certain changes to be made to the design object at startup. This allows design-object-based configuration to be done using the filesystem. In particular, this can be used to set an initial admin password, which is needed to modify the design object further in a running Cordra. See repolInit.json for general use of repolInit.json, and Locally Run Instance for the specific case of the admin password.

There are a number of options available for deploying Cordra. The simplest is a standalone Cordra installation that uses the local filesystem for storage and an embedded indexer. Cordra is configured to run in this mode by default, and it is a good setup for testing Cordra and any applications built using Cordra.

In addition to configuring Cordra software to use the local file system and embedded indexer, you can also configure a standalone Cordra installation to use external storage and/or indexing services. Those services would need to be setup independently, and then have Cordra configured to interact with them. Cordra currently supports the following backend services:

- Indexing: Apache Lucene (default), system memory Lucene, Elasticsearch, and Apache Solr. Click [here](#) for details.
- Storage: Filesystem (default), system memory, MongoDB, and Amazon S3. Click [here](#) for details.

Finally, multiple instances of Cordra can be configured as load-sharing nodes of an application, using external storage and indexing systems. This setup requires the use of Apache ZooKeeper and Apache Kafka to handle coordination between the nodes. For detailed instructions on setting up a distributed Cordra service, see Deploying Cordra as a Distributed System.
Management of complex infrastructure requires tools and tutorials related to keys management, distributed sessions management, logs management, user management, administrative interface, import-export tool, and environment migration.

### 5.2 Deploying Single-Instance Cordra

#### In This Section

- Software Distribution Details
- Locally Run Instance
  - Deploying Using Apache Tomcat
    * Configuring Tomcat for GraalVM JavaScript Compilation
- Docker Instance
  - Pre-configured Cordra Distribution
  - Deploying using Docker Compose
- Additional Configuration
  - Production Considerations
  - External Indexing and Storage

#### 5.2.1 Software Distribution Details

The Cordra software can be downloaded from [here](#). There are two options available: One which is pre-configured to use our hosted identifier (aka handle) service for a trial that allows digital objects to be locatable in the Internet, and one which you can configure to use your own handle server at a later point when you deem it is important for your users to auto locate digital objects. For more information about the Handle technology and setting up a handle server, please see [here](#).

For most users evaluating Cordra, we recommend trialing the hosted handle service by requesting a pre-configured Cordra software distribution. See [here](#) for details.

**Note:** The pre-configured Cordra software distribution includes a handle prefix and a necessary key pair useful for a quick start. We strongly recommend that you update the key pair if security is important during the trial.
5.2.2 Locally Run Instance

Before starting Cordra for the first time, the admin password needs to be set in a repoInit.json file under the data directory. This file should also contain the desired handle prefix, from which the unique identifiers will be generated; if the prefix is omitted, the prefix test will be used by default to create objects.

Pre-configured Cordra software distribution downloaded from the Cordra website includes the prefix, but admin password must be set prior to starting Cordra for the first time.

Example data/repoInit.json:

```json
{
    "adminPassword": "changeit",
    "prefix": "20.5000.123.TEST"
}
```

Once data/repoInit.json is in place, you can start Cordra with the startup script, and stop it with the shutdown script.

When Cordra starts, the admin password will be securely stored in the system and the repoInit.json file will be deleted. The admin password can then be changed using the UI while signed in as admin.

Cordra can be started by running the startup script in the distribution directory. The Cordra UI will be available at http://localhost:8080/ and https://localhost:8443. You can sign in as admin in order to further configure your Cordra server, if desired.

Except on Java 8, the GraalVM JavaScript engine used by JavaScript lifecycle hooks and methods supports compilation for improved performance. Certain JAR files for GraalVM compilation are not distributed with Cordra. To acquire these JAR files, you can use the bin/fetch-graalvm-compiler-jars script.

Cordra can by stopped by running the shutdown script.

5.2.2.1 Deploying Using Apache Tomcat

The Cordra software can be run using a servlet container instead of running in a standalone fashion as described above. The details below describe how to run Cordra using Apache Tomcat.

The Cordra war file is located in the Cordra distribution at sw/webapps-priority/cordra.war. It can be renamed to ROOT.war, for instance, if desired. (Note that sw/webapps-priority/ROOT.war in the Cordra distribution is included simply to forward requests to cordra.war by default.)

You should copy the war file into the Tomcat webapps directory,

The only change needed from a default Tomcat installation is to specify the Cordra data directory. This is done using Java system properties, which can be set in Tomcat via an environment variable CATALINA_OPTS. This setting can be set using the file bin/setenv.sh which can be created as a sibling of bin/catalina.sh. Ensure that bin/setenv.sh has the following contents:
CATALINA_OPTS="-Dcordra.data=/path/to/cordra/datadir ${CATALINA_OPTS}"

CATALINA_OPTS can also be used to specify memory configuration, such as:

CATALINA_OPTS="-Xmx2G -Dcordra.data=/path/to/cordra/datadir ${CATALINA_OPTS}"

Logging can be configured with a log4j2.xml file in the cordra.data directory.

Note: we recommend adding the following attributes to the Connector configuration in server.xml:

relaxedPathChars="[\]{{}^#x5c;\#x60;\&lt;&amp;gt;" relaxedQueryChars="[\]{{}^#x5c;\#x60;\&amp;gt;"  

This prevents Tomcat from giving errors if certain characters are used unencoded in URLs. Additionally we recommend in catalina.properties:

org.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
org.apache.catalina.connector.CoyoteAdapter.ALLOW_BACKSLASH=true

**Configuring Tomcat for GraalVM JavaScript Compilation**

Except on Java 8, the GraalVM JavaScript engine used by JavaScript lifecycle hooks and methods supports compilation for improved performance. To enable this on Tomcat, copy the files from the Cordra distribution in sw/lib/graalvm, and also the non-distributed files which can be downloaded using the script bin/fetch-graalvm-compiler-jars, to a directory accessible to Tomcat; in the following we assume they are in a directory graalvm which is a sibling of webapps. Then ensure that bin/setenv.sh has the following contents (in addition to the setting of cordra.data described above):

CATALINA_OPTS="-XX:+UnlockExperimentalVMOptions -XX:+EnableJVMCI --upgrade-module-path=$CATALINA_BASE/graalvm ${CATALINA_OPTS}"

Finally add to conf/context.xml, under <Context>:

<Loader delegate="true"/>

(If for some reason adding the Loader configuration is not desired, an alternative is to remove those JAR files from the Cordra webapp’s WEB-INF/lib directory which are also found in graalvm.)

**5.2.3 Docker Instance**

The Cordra distribution comes with the files necessary to build a Docker image that includes the Cordra software distribution.

There are four environment variables that can be set for the Docker image:

- CORDRA_ADMIN_PASS (REQUIRED) - password for the admin user
- CORDRA_BASE_URI - base uri for this Cordra HTTP API endpoint
- **CORDRA_PREFIX** - Handle prefix to use
- **CORDRA_HDL_ADMIN** - Handle admin for the prefix

These variables can be set on the command line, or in the `variables.env` file located in the `docker` folder.

To build the Docker image, use the following command:

```bash
docker build -t cordra -f docker/Dockerfile .
```

You can start Cordra using the following command:

```bash
```

To persist the Cordra data directory, the following command can be used. Note that the `$(pwd)/data` directory must exist in order for this command to work:

```bash
docker run -it -p8080:8080 -p8443:8443 -p2641:2641 -p9000:9000 \
--env-file docker/variables.env \
--mount type=bind,source="$(pwd)/data",target=/opt/cordra/data \
cordra
```

### 5.2.3.1 Pre-configured Cordra Distribution

If you have downloaded a pre-configured Cordra distribution from the Cordra website, the package will include `variables.env` file with the **CORDRA_PREFIX** and **CORDRA_HDL_ADMIN** variables populated. You should edit that file to fill in values for other environment variables including the admin password.

You can then build and run the Docker image as described above.

### 5.2.3.2 Deploying using Docker Compose

You may also use Docker compose to modify the Docker image built using the above instructions further. This is useful if you need to include the key pair your Cordra will use, or if you would like to modify the `config.json` file. There is a sample `docker-compose.yml` configuration file included that shows how to start a single instance of Cordra.

You can start Cordra by running this command from inside the `docker` subdirectory:

```bash
docker-compose up
```
5.2.4 Additional Configuration

There are a number of configuration options available for deploying Cordra. To learn more, see here.

5.2.4.1 Production Considerations

To deploy a single-instance Cordra in for productions, you will want to configure Cordra as a service. You will also likely want to put a reverse proxy in front of Cordra. For an example of how such a deployment might look, see Production Single Instance Deployment.

5.2.4.2 External Indexing and Storage

Cordra can be configured to use external indexing and storage services by editing the data/config.json file in the Cordra distribution. An example might look like this:

```json
{
    "httpPort": 8080,
    "listenAddress": "0.0.0.0",

    "index": {
        "module": "elasticsearch",
        "options": {
            "address": "localhost",
            "port": "9200",
            "addressScheme": "http",
            "index.mapping.total_fields.limit": "2000"
        }
    }

    "storage": {
        "module": "s3",
        "options": {
            "accessKey": "XXXXXXXXXXXXX",
            "secretKey": "XXXXXXXXXXXXX",
            "bucketName": "test.cordra.org",
            "s3KeyPrefix": "testing1234",
            "region": "us-east-1"
        }
    }
}
```

For more details on configuring external storage and indexing services, see Configuring Storage Backend and Configuring Indexing Backend.
5.3 Production Single Instance Deployment

This document describes an approach that could be used to set up a single instance of Cordra for production. This guide is written using an Ubuntu or Debian operating system. The commands will need to be adjusted for other systems.

5.3.1 Software Installation and Prerequisites

You will need Java 8+, Nginx, and unzip. You can install these with the following commands:

```
sudo apt update
sudo apt install unzip openjdk-11-jdk-headless nginx
```

You will also need to install certbot for managing HTTPS certificates. Please follow the instructions located on the Certbot website. Only follow the instructions through installing the software.

Download the Cordra distribution zip from https://www.cordra.org and save it on the server. Extract the zip file to `/opt/cordra/`.

Create a `cordra` user and make sure they have full access to the install directory.

You will also need a domain configured to point at the public IP address on your server. Setting up such a domain is outside of the scope of this document.

Finally, make sure ports 80, 443, 2641, and 9000 are publicly accessible through any firewall or security measures.

5.3.2 Configuration

1. Create an Nginx config file for cordra at `/etc/nginx/sites-available/cordra`. Replace `cordra.example.org` in the config file with your domain:

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    server_name cordra.example.org;

    access_log /var/log/nginx/cordra.access.log;

    location / {
        proxy_pass https://localhost:8443;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```
2. Enable the Cordra site and disable the default site:

```bash
sudo ln -s /etc/nginx/sites-available/cordra /etc/nginx/sites-enabled/
sudo rm /etc/nginx/sites-enabled/default
sudo systemctl reload nginx
```

3. Enable HTTPS with a Let's Encrypt certificate. Replace cordra.example.org with your domain:

```bash
sudo certbot --nginx -d cordra.example.org
```

4. In Cordra's data directory, configure the admin password using repoInit.json. Leave the defaults in config.json. Specifically, be sure to listen on localhost only, using port 8443. If you need to change the port, update the Nginx config as appropriate.

5. Start Cordra using the normal startup.sh script to test that everything is working. Shutdown using shutdown.sh.

6. Create `/lib/systemd/system/cordra.service` which looks like the following. Edit the user, Group, and paths as appropriate to your installation:

```
[Unit]
Description=Cordra Service
After=network.target

[Service]
Type=simple
User=cordra
Group=cordra
WorkingDirectory=/opt/cordra
RestartSec=10
Restart=always
StandardOutput=file:/opt/cordra/data/logs/service.log
StandardError=file:/opt/cordra/data/logs/service.log
ExecStart=/opt/cordra/startup
ExecStop=/opt/cordra/shutdown

[Install]
WantedBy=multi-user.target
```

7. Enable the Cordra service and start Cordra:

```bash
sudo ln -s /lib/systemd/system/cordra.service /etc/systemd/system/
sudo systemctl enable cordra
sudo systemctl start cordra
```
5.4 Deploying Cordra as a Distributed System

There are a number of options available for deploying Cordra. You can learn about those options here. Cordra can be deployed as a distributed system, with multiple instances of Cordra configured to work together as load-sharing nodes. For deployment in this manner, Cordra requires the use of those storage and indexing technologies that can be shared by all Cordra nodes simultaneously. This setup also requires the use of Apache ZooKeeper and Apache Kafka to handle coordination between the Cordra nodes. We describe below the steps needed to deploy Cordra in this distributed fashion.

5.4.1 Outline of Steps

1. Setup Apache ZooKeeper and Apache Kafka.
2. Setup shared storage, indexing, and sessions management.
3. Install and configure Cordra nodes using Apache Tomcat.
5. Load Cordra configuration information into ZooKeeper.

5.4.2 ZooKeeper and Kafka Configuration

The configuration files used by the various Cordra nodes are stored in ZooKeeper. Before installing and configuring Cordra, you will need to set up a ZooKeeper cluster followed by a Kafka cluster. Instructions for doing so are outside the scope of this document, but one setting in Kafka need customization:
5.4.3 Shared Backend Services

Distributed Cordra requires backend services that can be shared by multiple Cordra nodes. Furthermore, HTTP sessions must be also be shared. Click on the following links for details on the configuration needed for each option:

- Configuring Storage Backend
- Configuring Indexing Backend
- Distributed Sessions Manager

5.4.4 Apache Tomcat Configuration

Our recommended configuration for setting up Cordra as a distributed system is to use the servlet container Apache Tomcat, but any servlet container may be used. You can download and use any version of Tomcat up to v9.0.*; we have tested with Tomcat v8.5 and Tomcat v9.0.

If your infrastructure setup includes other applications that also need Tomcat, we recommend separate Tomcat instances for Cordra and any additional applications.

The Cordra war file, which should be inserted into the Tomcat webapps directory, is located in the Cordra distribution at sw/webapps-priority/cordra.war. It can be renamed to ROOT.war, for instance, if desired. (Note that sw/webapps-priority/ROOT.war in the Cordra distribution is included simply to forward requests to cordra.war by default.)

The main Tomcat configuration file is conf/server.xml. Summarizing the changes needed from the default configuration:

1. Change the shutdown port attribute of the outermost Server element, so that each Tomcat instance on one machine has a different shutdown port.

2. Set an address and port on the HTTP Connector element.

3. Configure an HTTPS Connector if needed.

4. Delete the AJP Connector (it is not needed).

5. It is recommended to use a distributed store for sessions, such as MongoDB, configured in Cordra as described in Distributed Sessions Manager. If instead Tomcat session replication is used, add a Cluster element to the Engine element as described in Session Replication. (Another alternative, instead of using a distributed session store or configuring session replication, is to place the Cordra instances behind a load balancer with "session affinity" or "sticky sessions".)

Note: we recommend adding the following attributes to the Connector configuration in server.xml:

```
relaxedPathChars="[]|{}|^#x5c;|^#x60;&quot;&amp;&lt;" relaxedQueryChars="[]|{}|^#x5c;|^#x60;&quot;&lt;"
```

This prevents Tomcat from giving errors if certain characters are used unencoded in URLs. Additionally we recommend in catalina.properties:
One other piece of configuration is needed to ensure that Cordra knows where to produce log files. This is done using Java system properties, which can be set in Tomcat via an environment variable CATALINA_OPTS. This setting can be set using the file bin/setenv.sh which can be created as a sibling of bin/catalina.sh. Ensure that bin/setenv.sh has the following contents:

```
CATALINA_OPTS="-Dcordra.data=/path/to/cordra/datadir ${CATALINA_OPTS}"
```

CATALINA_OPTS can also be used to specify memory configuration, such as:

```
CATALINA_OPTS="-Xmx2G -Dcordra.data=/path/to/cordra/datadir ${CATALINA_OPTS}"
```

Logging can be configured with a log4j2.xml file in the cordra.data directory.

Cordra log files can be forwarded to a separate indexing system as discussed here Managing Cordra Logs.

Additionally please see the configuration needed to ensure GraalVM JavaScript compilation at Configuring Tomcat for GraalVM JavaScript Compilation.

### 5.4.4.1 Session Replication

Note: the recommended approach to sessions in distributed Cordra is to use a distributed session store such as MongoDB, as described in Distributed Sessions Manager. This section describes an alternative approach using Tomcat's built-in session replication mechanism.

Tomcat session replication may be configured for each distributed set of Tomcat servers. This allows all the Tomcat instances to share authentication tokens.

If preferred, instead of configuring session replication, the Cordra instances can be placed behind a load balancer with “session affinity” or “sticky sessions”.

If configuring Tomcat for session replication, here is a sample of the relevant session replication configuration, inside the Engine element in Tomcat's server.xml:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelStartOptions="3" channelSendOptions="6">
  <Manager className="org.apache.catalina.ha.session.DeltaManager"/>
  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
              address="10.5.0.106"
              port="6000"
    />
    <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
      <Transport className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
    </Sender>
  </Channel>
</Cluster>
```
Note that in Tomcat 8.0, `MessageDispatchInterceptor` should be replaced with `MessageDispatch15Interceptor`.

Most of the above configuration snippet is boilerplate. The only things to change are the address/port attributes of the Receiver element, which correspond to the server being configured, and the host/port attributes of the Member elements, which correspond to the other servers in the group. Note that the server being configured should not be included in a Member element (and it is commented out in the above example).
5.4.4.2 Application Configuration

Cordra must be configured with ZooKeeper connection information, which indicates how to establish a connection with the configured ZooKeeper cluster. The rest of the configuration will be obtained from ZooKeeper.

ZooKeeper configuration can be placed in WEB-INF/web.xml using the zookeeperConnectionString and, optionally, configName context parameters. For example:

```xml
<context-param>
    <param-name>zookeeperConnectionString</param-name>
    <param-value>10.5.0.101:2181,10.5.0.102:2181,10.5.0.103:2181/cordra</param-value>
</context-param>
<context-param>
    <param-name>configName</param-name>
    <param-value>/config.json</param-value>
</context-param>
```

Note that in this case the configName parameter is the default /config.json and so could be omitted. See Cordra Configuration for details about the configuration.

These parameters can also be set as Java system properties (cordra.zookeeperConnectionString and cordra.configName) or environment variables (cordra_zookeeper_connection_string and cordra_config_name). For details on configuring Cordra to use TLS when accessing ZooKeeper, see Enabling TLS.

5.4.5 Load Balancer Configuration

You will need to set up a load balancer in front of the Tomcat servers hosting Cordra. If you are using Amazon EC2 to host your servers, create “Classic” Load Balancer for Cordra. Applications should be configured to talk to the Cordra load balancer, instead of talking to any given Cordra server directly.

If you have elected not to configure session replication in Tomcat, or configure Cordra itself to be responsible for distributed sessions (Distributed Sessions Manager), it will be necessary to configure the load balancer to use “session affinity” or “sticky sessions”, so that a client which was initially redirected by the load balancer to a particular Cordra node will continue to be forwarded to the same node for subsequent requests (and therefore the session information provided by the client continues to be accepted by the (only) Cordra node which knows that information).

5.4.6 Cordra Configuration

The Cordra config.json file should be stored as znode (on ZooKeeper) /cordra/config.json. If used, the private key should be stored as znode /cordra/privatekey, or, because the ZooKeeper zkCli.sh script does not provide an easy way to work with binary files, it can be stored in a Base64-encoded form as znode /cordra/privatekey.base64. If a Cordra repoInit.json file is used, it should be stored as znode /cordra/repoInit.json.

ZooKeeper includes a tool called zkCli.sh which can be used to install these files:
An individual Cordra instance can be given a different znode for configuration. This is done using the optional `configName` context-param in `web.xml`:

```xml
<context-param>
    <param-name>configName</param-name>
    <param-value>read-only-config.json</param-value>
</context-param>
```

The `configName` is interpreted relative to the `zookeeperConnectionString`, so this example would be znode `/cordra/read-only-config.json`.

Cordra keeps track of user requests that require re-processing as part of its fault tolerance logic. A transaction reprocessing queue is managed with the help the configured Kafka service tracked with the topic name `CordraReprocessing`. To boost performance benefits, you can enable multiple Cordra nodes to be able to process the transactions on the queue concurrently; for that, this Kafka topic should be created on Kafka with as many partitions as there are Cordra servers.

**Sample Cordra config.json:**

```json
{
    "isReadOnly": false,
    "index": {
        "module": "solr",
        "options": {
            "zkHosts": "10.7.1.101:2181,10.7.2.101:2181,10.7.3.101:2181/solr"
        }
    },
    "storage": {
        "module": "mongodb",
        "options": {
            "connectionUri": "mongodb://10.7.1.102:27017,10.7.2.102:27017,10.7.3.102:27017/replicaSet=rs0&w=majority&journal=true&wtimeoutMS=30000&readConcernLevel=linearizable"
        }
    },
    "reprocessingQueue": {
        "type": "kafka",
        "kafkaBootstrapServers": "10.7.1.101:9092,10.7.2.101:9092,10.7.3.101:9092"
    }
}
```

Note that Cordra's `config.json` no longer needs Cordra's listen address or port information, as that is now part of Tomcat configuration.
5.5 Reindexing

Occasionally it is useful to cause Cordra to recreate its index from its storage. For example, if externally managed Solr or Elasticsearch configuration is changed, then a reindex will cause objects to be searchable according to the new configuration.

The procedure to cause a reindex is:

1. Shut down all instances of Cordra.
2. Delete all objects from the index. For Lucene, this can be done by deleting the directory “cordraIndex” in the Cordra data directory. For Solr or Elasticsearch, this can be done through the index's own administrative API or UI.

Some configuration options for reindexing can be changed in Cordra's config.json, in a subobject “reindexing”:

```
{
   "reindexing": {
      "numThreads": 32,
      "batchSize": 16,
      "logChunkSize": 10000,
      "logProgressToConsole": false
   }
}
```

The values given are the defaults.

- “numThreads” specifies the number of threads used for reindexing.
- “batchSize" specifies the number of objects indexed per individual request to the indexing service. Together, “numThreads" and “batchSize" can be tuned for performance.
- “logChunkSize" specifies how frequently reindexing progress should be logged.
- “logProgressToConsole": if true, reindexing progress will be logged to the console in addition to Cordra's error.log.

By default Cordra will reindex all objects before startup is completed. In some applications, it may be preferred to index only certain necessary types first, and in some cases, to allow startup to complete, and then reindex remaining objects in the background. Search will be degraded during the background index, returning incomplete results.

To require objects of certain types to be indexed first, the property “priorityTypes" can be set in Cordra's config.json under "reindexing”. For example:

```
{
   "reindexing": {
   }
}
```

(continues on next page)
Types, users, and groups should generally be reindexed synchronously.

If references to JavaScript modules using types like "JavaScriptDirectory" are used (see Using External Modules), it is currently necessary to ensure that objects of type "Schema" and any sources of JavaScript (like type “JavaScriptDirectory”) are indexed first, using priorityTypes.

To enable background reindexing of other types, the boolean property "async" can be set. For example:

```json
{
    "reindexing": {
        "priorityTypes": [ "Schema", "User", "Group", "JavaScriptDirectory" ],
        "async": true,
        "isReadOnly": false,
        "index": {
            "module": "solr",
            ...
        }
    }
}
```

If async is enabled, then by default, Cordra will lock objects during the background reindex of non-priority types. This prevents an object from being updated at the same moment it is reindexed, which is needed to ensure that storage and index do not become out-of-sync. In some situations, though, this is not necessary; for example, when Cordra objects are only created and never updated, or when Cordra is only accessible read-only during the reindex. In such cases the performance of the background reindex can be improved by setting “lockDuringBackgroundReindex”: false:

```json
{
    "reindexing": {
        "priorityTypes": [ "Schema", "User", "Group", "JavaScriptDirectory" ],
        "async": true,
        "isReadOnly": false,
        "lockDuringBackgroundReindex": false
    }
}
```

For optimal performance, we also recommend increasing the “batchSize” to 100, when “lockDuringBackgroundReindex”:
5.6 Managing Cordra Logs

5.6.1 Access Logs

If Cordra is run within Tomcat, then Tomcat's default settings enable Tomcat to produce access logs. If Cordra is run as a standalone process, then Jetty is used for running Cordra. By default, Jetty does not produce access logs. The following sample configuration can be stored in data/jetty.xml for Jetty to produce access logs, where data is the Cordra's data directory:

```xml
<?xml version="1.0"?>

<Configure id="Server" class="org.eclipse.jetty.server.Server">
  <Set name="RequestLog">
    <New id="RequestLog" class="org.eclipse.jetty.server.AsyncNCSARequestLog">
      <Set name="filename"><SystemProperty name="cordra.data"/>/logs/jetty-request.log-yyyy_MM_dd</Set>
      <Set name="filenameDateFormat">yyyyMM</Set>
      <Set name="retainDays">0</Set>
      <Set name="append">true</Set>
      <Set name="extended">false</Set>
      <Set name="logCookies">false</Set>
      <Set name="LogTimeZone">GMT</Set>
      <Set name="LogLatency">false</Set>
      <!-- Uncomment the following to ignore paths used by Cordra UI -->
      <!--
      <Set name="ignorePaths">
        <Array type="java.lang.String">
          <Item>/js/*</Item>
          <Item>/lib/*</Item>
          <Item>/css/*</Item>
          <Item>/img/*</Item>
          <Item>/check-credentials</Item>
          <Item>/initData</Item>
          <Item>/schemaTemplates</Item>
          <Item>/updateHandles</Item>
        </Array>
      </Set>
      -->
    </New>
  </Set>
</Configure>
```
5.6.2 Error Logs

Cordra logs errors by default, and those entries are written to files that begin with `error.log` in the directory `data/logs`. One file per month will be produced, and the actual file name will reflect the year and month, e.g., `error.log-201510`, for the error log that corresponds to the entries for October 2015.

5.6.3 Index of Log

It may be useful to send the logged entries from access and error logs to an indexing service for retrieval and analysis. You may choose to use Elasticsearch as that indexing service. You will also need to install and configure two tools, Logstash and Filebeat to transform and register the logs with Elasticsearch.

The following instructions describe how to configure Logstash and Filebeat to register error logs with an Elasticsearch indexer that is running on port 9200 on localhost.

5.6.3.1 Install and Configure Logstash

1. **Install Logstash** using the instructions on the Logstash website.

2. Use the `logstash-plugin` command to install the following plugins:
   - `logstash-input-beats`
   - `logstash-output-elasticsearch`
   - `logstash-filter-date`
   - `logstash-filter-grok`
   - `logstash-filter-mutate`

3. Create a file called `cordra.conf` in the Logstash configuration directory. Copy the following into that file:

```ruby
input {
  beats {
    port => 5044
  }
}

filter {
  if [fields][app_source] == "cordra_error" {
    grok {
      match => [ "message", "%{DATA:[@metadata][timestamp]\s*\[%{DATA:thread}\]\s*%{LOGLEVEL:loglevel}\s*%{DATA:logger}\s*-%{GREEDYDATA:msg}" ]
    }
    date { match => [ "[@metadata][timestamp]", "yyyy-MM-dd HH:mm:ss.SSSZ" ]
  }
}
```
mutate { remove_field => ["message"] }

output {
  elasticsearch {
    hosts => ["localhost:9200"]
    # Use the following format for AWS hosted Elasticsearch
    # hosts => ["https://amazon-hosted-elasticsearch.example.com:443"]
    manage_template => false
    index => "%[@metadata][beat]-%{+YYYY.MM.dd}"
    document_type => "%[@metadata][type]"
  }
}

Be sure to edit the Elasticsearch host as necessary.

4. Start the Logstash service.

5.6.3.2 Install and Configure Filebeat

1. Install Filebeat using the instructions on the Filebeat website.

2. Manually load the Filebeat index template into Elasticsearch.

3. Replace the default filebeat.yml with the following:

```yaml
#================== Filebeat prospectors ==================

filebeat.prospectors:

- input_type: log

paths:
    - /path/to/cordra/logs/error.log*
exclude_files: ['\.gz$']

fields:
    app_source: cordra_error

# Java stack traces
multiline.pattern: '^[[:space:]]+\|Caused by:\'
multiline.pattern: '^[0-9]{4}-[0-9]{2}-[0-9]{2}'
multiline.negate: true
multiline.match: after
```

(continues on next page)
#====================== General ====================
name: cordra

#====================== Outputs ====================
output.logstash:
  hosts: ["127.0.0.1:5044"]

Edit Logstash host and log file location as necessary.

4. Start the Filebeat service.

## 5.7 Configuring Storage Backend

By default, Cordra uses the local file system for storing digital objects. However, Cordra can be configured to use alternate storage backend systems. It is mandatory to use an alternate storage backend system when Cordra is deployed as a distributed system.

There are a few storage technologies that Cordra can use for its storage. Cordra includes storage modules, which translate Cordra storage requirements into what each of the storage technologies natively offer.

To configure a storage module, other than for the default file system based storage, add a storage section to the Cordra `config.json` file. For example:

```
"storage" : {
  "module" : "module-name-goes-here",
  "options" : {

  }
}
```
5.7.1 Storage Modules

The following storage modules are included within the Cordra distribution:

5.7.1.1 Filesystem Storage

If no storage module is configured in `config.json`, the Cordra will store most information from digital objects in a local BerkeleyDB database and the payloads from those digital objects in a directory on the local filesystem. The identifiers of the payloads are hashed, and the hashes are used in the storage directory to ensure that payloads are spread out evenly among the storage sub-directories.

This storage module is only applicable for a single instance deployment scenario.

In the unusual case of needing explicit configuration (for example, with `Multiple Storages`, described below), use:

```
Module Name
bdbje
```

Module Options:

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cordra-DataDir</td>
<td>Filesystem path to the Cordra data directory; generally this will be automatically populated.</td>
</tr>
</tbody>
</table>

5.7.1.2 System Memory Storage

```
Module Name
memory
```

Module Options: none

The memory storage module uses the system memory; as such the digital objects will be erased once the Cordra process configured to use the memory module stops. Cordra configured with memory module is useful for testing purposes. There are no options required to be specified for this storage module.

This storage module is only applicable for a single instance deployment scenario.

The storage section of the `config.json` file looks like this:

```
"storage" : {
   "module" : "memory"
}
```
5.7.1.3 MongoDB Storage

**Module Name**

mongodb

**Module Options:**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionUri</td>
<td>MongoDB-style connection URI</td>
</tr>
<tr>
<td>maxTimeMs</td>
<td>“maxTimeMs” value used for MongoDB operations, which gives a time limit for processing; default is 30 seconds. Generally this should not need to be set.</td>
</tr>
<tr>
<td>maxTimeMsLongRunning</td>
<td>“maxTimeMs” value used for long-running MongoDB operations used in reindexing; default is 10 days. Generally this should not need to be set.</td>
</tr>
<tr>
<td>databaseName</td>
<td>Database name; defaults to “cordra”</td>
</tr>
<tr>
<td>collectionName</td>
<td>Collection name; defaults to “cordra”</td>
</tr>
<tr>
<td>gridFsBucketName</td>
<td>GridFS bucket name (for payload storage); defaults to “fs”</td>
</tr>
<tr>
<td>useLegacyIdField</td>
<td><em>(Advanced option.)</em> Boolean if set to false object ids will use the standard MongoDB property &quot;.id&quot; rather that a custom &quot;id&quot; property and index; defaults to true. Setting to the non-default false may have benefits for storage size or performance, but this cannot be changed for an existing storage.*</td>
</tr>
</tbody>
</table>

The MongoDB module will store objects in an MongoDB storage system. The `connectionUri` is a standard MongoDB-style connection string. If no `connectionUri` is configured, the default URI of `localhost:27017` will be used.

When connecting to MongoDB using TLS, additional configuration may be required. See *Enabling TLS* for details.
5.7.1.4 Amazon S3 Storage

Module Name
s3

Module Options:

<table>
<thead>
<tr>
<th>Option Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bucket-Name</td>
<td>(required) Name of bucket to use for storage.</td>
</tr>
<tr>
<td>region</td>
<td>(required) AWS region for bucket. (e.g., us-west-2)</td>
</tr>
<tr>
<td>accessKey</td>
<td>AWS access key for user with access to this bucket</td>
</tr>
<tr>
<td>secretKey</td>
<td>AWS secret key for user with access to this bucket</td>
</tr>
<tr>
<td>s3KeyPrefix</td>
<td>Prefix to use for keys on S3 objects.</td>
</tr>
<tr>
<td>endpoint</td>
<td>For setting a non-standard service endpoint.</td>
</tr>
<tr>
<td>signerOverride</td>
<td>Signature algorithm to be used by the client (possibly useful for non-standard service endpoints).</td>
</tr>
</tbody>
</table>

If accessKey and secretKey are omitted, credentials will be picked up from the environment as described here.

The S3 module stores Cordra objects in an Amazon S3 bucket. In order to use this module, you will need to create the bucket on AWS and create a user with full access to that bucket. An example is below:

```json
"storage" : {
    "module" : "s3",
    "options" : {
        "accessKey" : "XXXXXXXXXXXXXX",
        "secretKey" : "XXXXXXXXXXXXXX",
        "bucketName" : "my-bucket-name.example.org",
        "s3KeyPrefix": "testing1234",
        "region": "us-east-1"
    }
}
```

5.7.1.5 Multiple Storages

Module Name
multi

Module Options:
<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storageChooser</td>
<td>(required) Fully-qualified class name of an implementation of net.cnri.cordra.storage.multi.StorageChooser.</td>
</tr>
<tr>
<td>storageChooserOp-</td>
<td>(optional) A JSON object passed to the constructor of the StorageChooser.</td>
</tr>
<tr>
<td>tions</td>
<td></td>
</tr>
<tr>
<td>storageMap</td>
<td>(required) A map from String names of storages to storage configurations (with “module” and “options” properties).</td>
</tr>
</tbody>
</table>

This module allows multiplexing among several storage implementations. A custom implementation of net.cnri.cordra.storage.multi.StorageChooser can be provided to determine which storage is accessed for each call.

This module can be used if different types of digital objects are to be managed in different storage systems.

For a standard single-instance Cordra deployment, a JAR file containing the class can be placed in the lib subdirectory of the Cordra data directory, along with any dependency JARs (the cordra-core and Gson dependencies will be provided automatically). If Cordra is deployed in a separate servlet container, the JAR file should be deployed in the servlet container or in Cordra's own WEB-INF/lib directory.

The StorageChooser can make use of a special feature of the REST API: any call can take a query parameter “requestContext”, which encodes a JSON object. That user-supplied context is made available to the methods of the StorageChooser.

### 5.7.1.6 Custom Storage

**Module Name**

```
custom
```

It is possible to create a custom storage module which implements the Java interface net.cnri.cordra.storage.CordraStorage. In addition to "module": "custom", there should be a sibling property of "module", "className", which should be set to the fully-qualified name of the Java class implementing CordraStorage.

If the class has a constructor which takes a com.google.gson.JsonObject, the "options" from the configuration will be passed to that constructor to instantiate the class. If the "options" does not already have a property "cordraDataDir", that property will be populated with the filesystem path of the Cordra data directory. If there is no constructor taking a JsonObject, a default constructor (taking no arguments) will be called.

For a standard single-instance Cordra deployment, a JAR file containing the class can be placed in the lib subdirectory of the Cordra data directory, along with any dependency JARs (the cordra-core and Gson dependencies will be provided automatically). If Cordra is deployed in a separate servlet container, the JAR file should be deployed in the servlet container or in Cordra's own WEB-INF/lib directory.
5.8 Configuring Indexing Backend

Cordra uses indexers that are based on Apache Lucene, such as Lucene itself, Apache Solr, and Elasticsearch.

By default, Cordra uses Apache Lucene that is configured to use the local file system for storing the indexes. However, Cordra can be configured to use alternate indexing backend systems. It is mandatory to use an alternate indexing backend system when Cordra is deployed as a distributed system.

There are a few indexer technologies that Cordra can use for indexing. Cordra includes index modules, which translate Cordra indexing requirements into what each of the indexer technologies natively offer.

To configure a index module, other than for the default file system based index, add a `index` section to the Cordra `config.json` file. For example:

```json
"index" : {
  "module" : "module-name-goes-here",
  "options" : {

  }
}
```

The following index modules are included within the Cordra distribution.

### 5.8.1 Index Modules

There are currently four indexing backends supported by Cordra.

#### 5.8.1.1 Lucene Index

**Module Name:** lucene

**Module Options:**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowLeading-Wildcard</td>
<td>Allow queries to start with a wildcard (*). May have performance implications on large indexes. (Default: false).</td>
</tr>
</tbody>
</table>

If no indexing backend is configured in `config.json`, the Cordra will use a filesystem-based Apache Lucene indexer. This module is only applicable for a single instance deployment scenario.
5.8.1.2 System Memory Lucene Index

**Module Name:** memory

**Module Options:**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowLeading-Wildcard</td>
<td>Allow queries to start with a wildcard (*). May have performance implications on large indexes. (Default: <code>false</code>).</td>
</tr>
</tbody>
</table>

This module uses Lucene, but the index gets erased once the Cordra process is stopped. This module is useful for testing and is also only applicable for a single instance deployment scenario.

The `index` section of the `config.json` file looks like this:

```json
"index" : {
   "module" : "memory"
}
```

5.8.1.3 Elasticsearch Index

**Module Name:** elasticsearch

**Module Options:**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>address</td>
<td>Address of Elasticsearch server (Default: <code>localhost</code>)</td>
</tr>
<tr>
<td>addressScheme</td>
<td>Protocol for Elasticsearch server (Default: <code>https</code>)</td>
</tr>
<tr>
<td>port</td>
<td>Port number for Elasticsearch server (Default: <code>9200</code>)</td>
</tr>
<tr>
<td>baseUri</td>
<td>URI(s) of Elasticsearch server(s). If specified, this will be used instead of the previous address settings. Can be specified as a string or as an array and can be specified as &quot;baseUris&quot;.</td>
</tr>
<tr>
<td>indexName</td>
<td>Name of the index to use (Default: <code>cordra</code>)</td>
</tr>
<tr>
<td>authorization</td>
<td>If specified, the value of an Authorization: header to include with every request to Elasticsearch.</td>
</tr>
<tr>
<td>username, password</td>
<td>If specified and authorization is absent, a Basic Authorization header will be included with every request.</td>
</tr>
<tr>
<td>mappings</td>
<td>Mappings to be used when Cordra initializes the index, to augment or override Cordra defaults. If the index already exists, has no effect.</td>
</tr>
<tr>
<td>index.*</td>
<td>Index setting for Cordra index.</td>
</tr>
</tbody>
</table>
The Elasticsearch indexer works with both self-hosted instances of Elasticsearch and Amazon’s hosted Elasticsearch service. Cordra currently supports Elasticsearch versions 7 and 8.

By default, Cordra sets `index.mapping.total_fields.limit` to 10000. You can override this or send additional index configuration to Elasticsearch by including the appropriate `index.*` setting in your configuration. For example, to set the limit on total fields for the index to 5000, you could use the following configuration:

```json
"index" : {
   "module" : "elasticsearch",
   "options" : {
      "address" : "localhost",
      "port" : "9200",
      "addressScheme" : "http",
      "index.mapping.total_fields.limit": "5000"
   }
}
```

When connecting to Elasticsearch using TLS, additional configuration may be required. See *Enabling TLS* for details.

### 5.8.1.4 Solr

**Module Name:** solr

**Module Options:**

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseUri</td>
<td>URI of Cordra index on Solr indexing server. This should include the core name. (Default: <a href="http://localhost:8983/solr/cordra">http://localhost:8983/solr/cordra</a>)</td>
</tr>
<tr>
<td>zkHosts</td>
<td>Connection string for ZooKeeper cluster used with SolrCloud.</td>
</tr>
<tr>
<td>collectionName</td>
<td>Name of the collection to use with SolrCloud (Default: cordra)</td>
</tr>
<tr>
<td>minRf</td>
<td>A number for the minimum desired replication factor in a SolrCloud configuration. If the achieved replication is lower Cordra will log a warning. Generally this will be set automatically based on SolrCloud configuration in ZooKeeper; this option can be used to set it lower to prevent warnings when Solr nodes are known to be down.</td>
</tr>
</tbody>
</table>

Cordra can be configured to connect to a standalone Solr server or a Solr Cloud cluster with its configuration stored in ZooKeeper. Cordra currently supports Solr versions 6, 7, and 8.

In addition to the Solr setting in the Cordra `config.json` file, the following Solr configuration file updates must be made. The default managed-schema file (called `schemas.xml` on older versions of Solr) should be replaced with the following (which can be downloaded here: [link])

*Configuring Indexing Backend*
<?xml version="1.0" encoding="UTF-8" ?>
<schema name="cordra" version="1.6">
    <uniqueKey>id</uniqueKey>
    
    <fieldType name="long" class="solr.TrieLongField" precisionStep="0" positionIncrementGap="0" />
    <fieldType name="double" class="solr.TrieDoubleField" precisionStep="0" positionIncrementGap="0" />
    <fieldType name="id" class="solr.TextField">
        <analyzer>
            <tokenizer class="solr.KeywordTokenizerFactory" />
        </analyzer>
    </fieldType>
    
    <fieldType name="keyword" class="solr.TextField" positionIncrementGap="10000">
        <analyzer>
            <tokenizer class="solr.KeywordTokenizerFactory" />
            <filter class="solr.LowerCaseFilterFactory" />
        </analyzer>
    </fieldType>
    
    <fieldType name="text" class="solr.TextField" positionIncrementGap="10000">
        <analyzer>
            <tokenizer class="solr.StandardTokenizerFactory" />
            <filter class="solr.LowerCaseFilterFactory" />
        </analyzer>
    </fieldType>
    
    <fieldType name="dateRangeField" class="solr.DateRangeField" />
    <fieldType name="datePointField" class="solr.DatePointField" />
    
    <field name="_version_" type="long" indexed="true" stored="true" multiValued="false" />
    <field name="id" type="id" indexed="true" stored="true" required="true" />
    <field name="repoid" type="keyword" indexed="true" stored="false" />
    <field name="type" type="keyword" indexed="true" stored="true" />
    <field name="aclRead" type="keyword" indexed="true" stored="false" multiValued="true" />
    <field name="aclWrite" type="keyword" indexed="true" stored="false" multiValued="true" />
    <field name="createdBy" type="keyword" indexed="true" stored="false" />
    <field name="remoteRepository" type="keyword" indexed="true" stored="false" />
    <field name="username" type="keyword" indexed="true" stored="false" />
    <field name="users" type="keyword" indexed="true" stored="false" multiValued="true" />
    <field name="schemaName" type="keyword" indexed="true" stored="false" />
    <field name="javaScriptModuleName" type="keyword" indexed="true" stored="false" multiValued="true" />
    <field name="isVersion" type="keyword" indexed="true" stored="false" />
</schema>

(continues on next page)
The default `solrconfig.xml` file should be modified in the following ways:

- Change the `maxTime` value of `autoSoftCommit` to 10000
- Change the `maxTime` value of `autoCommit` to 60000
- Make sure the `openSearcher` value of `autoCommit` is set to `false`
- Remove or comment out the `searchComponent` named "elevator" and the `requestHandler` named "/elevate"
- Remove or comment out the `updateRequestProcessorChain` named "add-unknown-fields-to-the-schema"
- Remove or comment out any `initParams` setting that make use of "add-unknown-fields-to-the-schema"
- In `initParams`, change the value of `df` to `internal.all`. Any other `df` values used should also be changed to `internal.all`.

Configuring Indexing Backend
An example of a fully modified solrconfig.xml can be downloaded here.

When connecting to Solr using TLS, additional configuration may be required. See Enabling TLS for details.

### 5.8.2 Phrase Queries

You should refer to the query syntax supported by the indexing backend system that you configured with your Cordra instance.

One point is worth noting here. Queries that are placed within double quotes trigger exact match searches. However, queries without double quotes will be tokenized in a way which can sometimes be surprising. This is a side effect of the tokenization used by Lucene, Solr, and Elasticsearch.

For example, suppose you send the query `/name:foo-bar` to the indexer. The value is tokenized and treated as an OR statement. The query becomes `/name:(foo bar)`, which will match items with the name foo, bar, foo-bar, and bar-foo. However, with double quotes, the query is turned into a phrase query, `/name:"foo-bar"`, which will only match items with the name “foo-bar”.

In general, you should ensure that double quotes are used when a search might result in multiple tokens and only matches of the entire phrase are desired.

### 5.9 Distributed Sessions Manager

By default, Cordra handles HTTP sessions with an in-memory sessions manager. However, Cordra can be configured to use an alternate sessions manager. To configure this, add a `sessions` section to the Cordra `config.json` file.

For example:

```json
"sessions" : {
    "module" : "module-name-goes-here",
    "options" : {

    }
}
```

If a property specifying whether the account is active is configured on the User digital object, you will be able to invalidate user sessions by deactivating the user account. See `"auth": "accountActive"` for details on how to configure account activation.
5.9.1 Session Managers

There are currently three session managers supported by Cordra, as defined below.

5.9.2 Common Options

All session managers take the following options:

<table>
<thead>
<tr>
<th>Option name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>module</td>
<td>The name of the module to use. Default is “memory”</td>
</tr>
<tr>
<td>timeout</td>
<td>Number of seconds before a session is expired. Default is 30 seconds</td>
</tr>
</tbody>
</table>

5.9.2.1 System Memory

Module Name
memory

If no sessions module is configured in config.json, Cordra will use the system memory sessions manager. This sessions manager does not persist sessions over a reboot. Session information stored in memory will be lost if Cordra is restarted.

These sessions are not distributed, and so can only be used in a single instance deployment scenario.

5.9.2.2 Apache Tomcat

Module Name
tomcat

With this configuration, Cordra will use the Tomcat's built-in facility for sessions. This is fine for a single-instance deployment, and can be used for a distributed deployment if Tomcat is configured for distributed sessions (see Session Replication for an example of this).

This should not be used with a default Cordra single-instance installation, which uses Jetty instead of Tomcat.

5.9.2.3 MongoDB

Module Name
mongodb

The MongoDB sessions manager will store sessions in an external MongoDB system. This module requires additional configuration under a property “options”. Under “options”, there must be a parameter called `connectionUri`, which is a standard MongoDB connection string. The “options” can also include “databaseName” (defaults to “cordra”) and “collectionName” (defaults to “sessions”). (If Cordra’s “sessions” and “storage” both use MongoDB, they should use distinct collections.)

For example:
5.10 Managing Users and Groups

Any regular object can be converted into a User or a Group object for purposes of ACLs by adding a special property `auth` to that object's schema. See here for details on that property. By converting an object into a User or a Group, Cordra can enable such objects (once authenticated) to exercise access control over other objects.

Objects that are converted into Users or Groups, as such, can be managed as regular objects using the API by any user with permission to do so.

Additionally, they can be created and modified in the Cordra UI by authorized users.

JavaScript based rules can be used on User and Group objects, as with any other object type. This can be useful for tasks like checking password strength, ensuring that usernames meet certain criteria, or preventing users or groups from being deleted unless certain conditions are met. See Example: User Schema JavaScript for an example of how to use JavaScript rules to perform validation and enrichment tasks.

See Authenticate Hook for the lifecycle hook allowing customization of authentication.

5.10.1 Adding Users

In the Cordra UI, click "Create" and choose the "User" type.

This will open the UI for editing the new User object.
The username and password should be filled in. If the user's password should be changed after the next authentication, set the "Require Password Change" field to true. Click the save button to save the User object.

5.10.2 Adding Groups

In the Cordra UI, click "Create" and choose the "Group" type.

This will open the UI for editing the new Group object.
The Group name should be filled in, but the description field is optional. When ready, click the Add User button to add identifiers of user objects.

In the user field, begin by typing the username of the user you would like to add. As potential matches are found, search results will pop up, allowing you to select the desired user.

Once the appropriate user has been found, click on the username to fill in the user field with the
User object identifier.

Repeat these steps for each user to be added to the group. When all users have been added, click the "Save" button to save the Group object

### 5.11 Administrative UI

Cordra includes a graphical interface for web browsers which enables users to create, retrieve, update, delete, search, and protect digital objects. When logged into the Cordra UI as an administrator, you will see an additional "Admin" dropdown menu at the top of the screen.

Each of these options can be used to modify how Cordra looks and functions. The options are described below.
5.11.1 Types

The Types page can be used to add and/or modify the types stored in Cordra. A list of types is shown. There are three default types. To edit a type, click on its name. An editor will appear, showing the schema and JavaScript for the type. Make any changes you would like, and then click the “Save” button.

New types can be added either one at a time or in bulk. To add an individual type, click the “Add” button. In the dialog that pops up, choose a name for the type and a template. If you would like for an example JavaScript to be populated in the editor, select the “Include example JavaScript” checkbox. Then click the “Add” button. The dialog will close, and the new type will be available in the editing interface. Make any additional changes required, and then click the “Save” button to save the new type.

To add multiple types at once, you will first need JSON file containing the types. Here is an example file:

```json
{
  "results": [
    {
      "id": "test/171a0606f7c74580fd39",
      "type": "Schema",
      "content": {
        "identifier": "test/171a0606f7c74580fd39",
        "name": "Group",
        "schema": < Schema json omitted for brevity >,
        "javascript": < JavaScript omitted for brevity >
      },
      "metadata": {
        "createdOn": 1535479938849,
        "createdBy": "admin",
        "modifiedOn": 1535479938855,
        "modifiedBy": "admin",
        "txnId": 65
      }
    },
    {
      // contents omitted for brevity
    }
  ]
}
```

(continues on next page)
The format of the file is similar to the format of the response to an object query. You can download json for types currently in Cordra using the Search API:

```
GET /search?query=type:"Schema"
```

The results of this query can be edited to create a new file for upload. Extra fields like pageNum and pageSize do not need to be removed.

To upload the types file, first click the “Load from file” button in the Types admin UI. Next, select the file to upload. If you would like to delete existing types, check the checkbox indicating such. If you choose not to delete existing types first, an error will be throw if you try to upload a duplicate type. Click the “Load” button to load the types into Cordra.

### 5.11.2 Design Object

See Design Object for details.

### 5.11.3 UI Menu

Some UI configuration is stored as JSON within Cordra. Clicking on the UI menu will bring up an editor that can be used to modify this configuration.

Example UI Configuration:

```json
{
    "title": "RepositoryTest",
    "allowUserToSpecifySuffixOnCreate": false,
    "initialFragment": "urls/intro.html",
    "metadata": {
        "createdOn": 1535479938849,
        "createdBy": "admin",
        "modifiedOn": 1535479938855,
        "modifiedBy": "admin",
        "txnId": 65
    }
}
```
"relationshipsButtonText": "Show Relationships",
"navBarLinks": [
  {
    "type": "url",
    "title": "External link",
    "url": "https://example.com/
  },
  {
    "type": "about",
    "title": "About"
  },
  {
    "type": "menu",
    "title": "Queries",
    "links": [
      {
        "type": "query",
        "title": "All Objects",
        "query": "*:*",
        "sortFields": [
          {
            "name": "metadata/modifiedOn",
            "reverse": true
          }
        ]
      },
      {
        "type": "query",
        "title": "Documents createdBy user",
        "query": "+type:Document +(metadata/createdBy:"{0}") +(/
        
        "description:{1})",
        "queryForm": {
          "description": "Form to select user and specify words in /
        
        "description",
          "fields": [
            {
              "name": "Username",
              "type": "handleReference",
              "types": [
                "User"
              ]
            },
            {
              "name": "Words in description",
              "type": "string"
            }
          ]
        }
      }
    ]
  }
]
Here are the attributes available in the UI configuration object.

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title</td>
<td>The text used in the title bar to identify the service.</td>
</tr>
<tr>
<td>relationshipsButtonText</td>
<td>The text shown on the button used to show the relationships between objects.</td>
</tr>
<tr>
<td>allowUserToSpecifySuffixOnCreate</td>
<td>Provides an input box for the suffix of the object id when creating objects. The prefix is set in the handleMintingConfig.</td>
</tr>
<tr>
<td>allowUserToSpecifyHandleOnCreate</td>
<td>Provides an input box for the complete object id when creating objects.</td>
</tr>
<tr>
<td>initialQuery</td>
<td>A query to be loaded if none is present when the app is loaded.</td>
</tr>
<tr>
<td>initialFragment</td>
<td>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.</td>
</tr>
<tr>
<td>initialSortFields</td>
<td>Sort fields to use in the UI if none is specified.</td>
</tr>
<tr>
<td>hideTypeInObjectEditor</td>
<td>Do not display object type under object ID in UI when editing.</td>
</tr>
<tr>
<td>numTypesForCreateDropdown</td>
<td>Number of types to display in the creation dropdown. If more than this number are available, a search interface will be shown instead.</td>
</tr>
<tr>
<td>aclUiSearchTypes</td>
<td>A list of types to be used in the UI for editing ACLs; typically the list of types which represent users and groups.</td>
</tr>
<tr>
<td>navBarLinks</td>
<td>An array of objects used for adding links to the navigation bar. Details below.</td>
</tr>
<tr>
<td>searchResults</td>
<td>Configuration for search results. Details below.</td>
</tr>
<tr>
<td>customAuthentication</td>
<td>Configuration of custom UI for client interaction with a 3rd party authentication provider to acquire an access token. See External Authentication Provider for details.</td>
</tr>
</tbody>
</table>
5.11.3.1 navBarLinks

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Can be query, typeDropdown, typeObjectsDropdown, url, page, about, or menu.</td>
</tr>
<tr>
<td>title</td>
<td>The text used on the link.</td>
</tr>
<tr>
<td>query</td>
<td>If the type is query this attribute holds the query to run.</td>
</tr>
<tr>
<td>sortFields</td>
<td>Sort fields for the query results.</td>
</tr>
<tr>
<td>url</td>
<td>If the type is url this attribute holds the url.</td>
</tr>
<tr>
<td>location</td>
<td>If the type is page this attribute holds the location of the page.</td>
</tr>
<tr>
<td>links</td>
<td>If the type is menu this property contains an array of link object that should be on menu.</td>
</tr>
<tr>
<td>queryForm</td>
<td>If the type is query this property contains configuration for a form that can be used to populate a pattern query.</td>
</tr>
</tbody>
</table>

Link Type query:

```
{
    "type": "query",
    "title": "All Objects",
    "query": "*:*",
    "sortFields": [
        {
            "name": "metadata/modifiedOn",
            "reverse": true
        }
    ]
}
```

A link of type query can contain a query and sortFields configuration. When clicked on that query will be performed and the results shown in the UI. The above example searches for all objects sorted by most recently modified.

Link Type query with a queryForm:

```
{
    "type": "query",
    "title": "Documents createdBy user",
    "query": "+type:Document +(metadata/createdBy:"{0}") +(/description:{1})",
    "queryForm": {
        "description": "Form to select use and specify words in /description",
        "fields": [
            {
                "name": "Username",
                "type": "handleReference",
                "types": [
```
Sometimes it can be helpful to present a UI form for entering values that are then composed into a query. Here the query contains the numeric place holders {0} and {1}. The queryForm contains an array of fields. Each field will be rendered in a UI form for the user to interact with. When the query is run the values from those fields will be substituted into the numeric place holders in the query based on field order in the fields array. For example here the first field value will replace the {0} and the second field value will replace the {1}.

The above example uses string and handleReference field types. Available types are:

string: This will provide a text box in the form. The value of that text box will be substituted into the corresponding numeric place holder in the query.

boolean: This will provide a check box in the form. The value from the checkbox will be inserted into the query as the strings “true” or “false”

enum: Use this in combination with an array property allowedValues. The will provide a select box containing the values.

handleReference: This field type will create a search box in the form. The user can enter term to search for objects and select an object from the results. The id of the selected object will inserted into the queries numeric placeholder. Use this in combination with array properties types and optionally excludeTypes. The values in those arrays controls what types of objects may be selected.

Link Type url:

```json
{
  "type": "url",
  "title": "External link",
  "url": "https://example.com/
}
```

The url link when clicked on will show the page at the http url in an iframe embedded in the Cordra UI.

Link Type page:

```json
{
  "type": "page",
  "title": "Home",
}
```

(continues on next page)
The page link when clicked on will show the internal page at the given url. If the location attribute does not begin with a hash (#), one will be prepended.

Link Type about:

```
{
  "type": "about",
  "title": "About"
}
```

The about link will display a page showing the Cordra server version information.

Link Type typeDropdown:

```
{
  "type": "typeDropdown",
  "title": "Show Objects of Type",
  "maxItems": 15
}
```

The typeDropdown link will add a menu listing the names of the types in the system. Clicking on one of those names will search for all objects of that type.

Link Type typeObjectsDropdown:

```
{
  "type": "typeObjectsDropdown",
  "title": "Types",
  "maxItems": 15
}
```

The typeObjectsDropdown link will add a menu listing the names of the types in the system. Clicking on one of those names will show the Schema object with that name.

Link Type menu:

```
{
  "type": "menu",
  "title": "Queries",
  "links": [
    {
      "type": "query",
      "title": "All Objects",
      "query": "*:*",
      "sortFields": []
    }
  ]
}
```

(continues on next page)
Here we create a menu in the navigation bar that contains links to two queries.

### 5.11.3.2 searchResults

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeType</td>
<td>Include type in results display.</td>
</tr>
<tr>
<td>includeModifiedDate</td>
<td>Include modification date in results display.</td>
</tr>
<tr>
<td>includeCreatedDate</td>
<td>Include creation date in results display.</td>
</tr>
</tbody>
</table>

### 5.11.3.3 customAuthentication

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>Url of the custom html page to be inserted into an iframe in the authentication dialog.</td>
</tr>
<tr>
<td>tabName</td>
<td>Name of the tab to be for this custom authentication in the authentication dialog.</td>
</tr>
<tr>
<td>height</td>
<td>Optional height to set the iframe.</td>
</tr>
</tbody>
</table>

### 5.11.4 Authorization Menu

Default access control lists are stored as JSON within Cordra. Clicking on the Authorization menu will bring up an editor that can be used to modify this configuration.

Example Authorization Configuration:

```json
{
    "schemaAcls": {
        "User": {
            ...
        }
    }
}
```
For more information on configuring ACLs in Cordra, see Authorization.
5.11.5 Handle Records Menu

This screen can be used to modify Cordra's Handle minting configuration. Once the configuration is modified, you can use the "Update All Handles" button to propagate changes to the affects Handle records. For more information on configuring Cordra and Handle integration, see Handle Integration.

5.11.6 Security Menu

This screen can be used to reset the password for the built-in Cordra admin user.

5.11.7 Design JavaScript Menu

JavaScript can be added to the Design object for the purposes of programmatically generating object ids when creating new objects. see Generate Object Id Hook, as well as for customizing queries, see Customize Query Hook.

Additionally, service-level static type methods can be defined in the Design object JavaScript, in the same way they are defined on Type objects. See Service-level Static Methods for details.

**Warning:** Care must be taken when importing Type methods using require statements. Since the Design object is sometimes loaded before any Types it might be requiring from, top-level require statements may fail. In such cases, you may need to import the module from within the method instead.

5.12 Enabling TLS

In addition to responding over HTTPS, Cordra can be configured to use TLS/SSL when communicating with underlying systems.

See also CordraClient TLS configuration for how to configure the Java CordraClient for server TLS authentication when connecting to Cordra.

5.12.1 ZooKeeper

**Note:** TLS connections are only supported in ZooKeeper 3.5.5 or higher.

Connecting to ZooKeeper using TLS requires certain system properties to be set. These properties can also be set in the WEB-INF/web.xml file using a context-param called properties (in Java properties format) or properties.json (in JSON format). If using Java properties format note that whitespace at the end of a line, but not beginning of a line, is significant.

Example web.xml:
For more details on ZooKeeper client configuration, see the ZooKeeper Administrator’s Guide and the ZooKeeper SSL User Guide.

### 5.12.2 Kafka

Generally, the Kafka clients created by Cordra will use the same TLS configuration. This can be set using the `kafkaCommonConfig` property in `config.json`. Any configuration in this property will be applied to all internal Kafka clients, unless that setting is overridden by a specific client configuration.

Sample `config.json`:

```json
{
   "kafkaCommonConfig": {
      "security.protocol": "SSL",
      "ssl.key.password": "secret_key",
      "ssl.keystore.location": "/path/to/keystore",
      "ssl.keystore.password": "keystore_secret",
      "ssl.truststore.location": "/path/to/truststore",
      "ssl.truststore.password": "truststore_secret",
      "ssl.endpoint.identification.algorithm": "https"
   }
}
```

For more details on Kafka client configuration, see the Kafka documentation.
5.12.3 General TLS

Connecting to some applications may require modifying `javax.net.ssl` system properties. For example, if you are using mutual TLS authentication or using self-signed certificates, with MongoDB, Solr, or Elasticsearch, you may need to set the properties `javax.net.ssl.keyStore`, `javax.net.ssl.keyStorePassword`, `javax.net.ssl.trustStore`, and `javax.net.ssl.trustStorePassword`.

Cordra supports configuring system properties using the `properties` property in `config.json`. Example `config.json` with these TLS settings:

```json
{
   "properties": {
      "javax.net.ssl.keyStore": "/path/to/keystore",
      "javax.net.ssl.keyStorePassword": "keystore_secret",
      "javax.net.ssl.trustStore": "/path/to/truststore",
      "javax.net.ssl.trustStorePassword": "truststore_secret"
   }
}
```

5.13 Configuring HTTPS Keys

Cordra provides HTTP APIs, notably *DOIP API for HTTP Clients*, along with a graphical interface for web browsers. The HTTP endpoint for these interfaces is strongly recommended to be based on HTTPS. We describe below how to generate the keys necessary to enable HTTPS.

If deploying Cordra in a servlet container such as Tomcat, the container will control HTTPS configuration.

There are three methods for configuring HTTPS keys in standalone Cordra:

1. using automatically-generated, self-signed keys (default)
2. replacing the generated keys in the `data` directory with specific keys, including a browser-valid TLS certificate
3. configuring Cordra to use keys located elsewhere in the filesystem

The default behavior allows Cordra to serve HTTPS traffic, but users will see a browser warning about the self-signed key. To remove this warning, configure a key as described below.

5.13.1 Replacing Files in Default Location

If you have a signed set of keys, along with a certificate chain file, you can simply copy those files into your Cordra `data` directory, overwriting the generated keys. Copy the private key to `httpsPrivateKey.pem` and the certificate to `httpsCertificate.pem`. You will need to restart in order for Cordra to recognize the new keys.

Assuming your keys are correctly signed, you should now be able to view the Cordra UI without any errors or warnings displayed by the web browser.
5.13.2 Configure Key Location

Instead of copying your keys into Cordra, you can put information about the location of your keys on disk in the Cordra `config.json` file. An advantage to using this method to configure your keys is that Cordra will automatically reload the keys if the files are modified. This is useful if you are using keys that change frequently (for example, keys from Let's Encrypt).

5.13.2.1 PEM Files

To configure Cordra to use standard PEM-formatted files, add the following to parameters to the `config.json` file.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>httpsPrivKeyFile</code></td>
<td>required</td>
<td>Path to the private key file. Must be encoded in the PKCS#8 format.</td>
</tr>
<tr>
<td><code>httpsCertificateChainFile</code></td>
<td>required</td>
<td>Path to the certificate chain file.</td>
</tr>
<tr>
<td><code>httpsKeyPassword</code></td>
<td>optional</td>
<td>Password to use for decrypting the private key, if needed.</td>
</tr>
</tbody>
</table>

Example configuration:

```
{
  "httpPort": 8080,
  "httpsPort": 8443,
  "listenAddress": "0.0.0.0",
  "httpsPrivKeyFile": "/path/to/keys/example.com.key",
  "httpsCertificateChainFile": "/path/to/keys/example.com.crt"
}
```

5.13.2.2 KeyStore Files

You can also configure Cordra to use a keystore. Here are the parameters to add to the `config.json` file.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>httpsKeyStoreFile</code></td>
<td>required</td>
<td>Path to the keystore file.</td>
</tr>
<tr>
<td><code>httpsAlias</code></td>
<td>required</td>
<td>Alias for the keystore entry to use.</td>
</tr>
<tr>
<td><code>httpsKeyStorePassword</code></td>
<td>optional</td>
<td>Password for the keystore. One of this or <code>httpsKeyPassword</code> is required.</td>
</tr>
<tr>
<td><code>httpsKeyPassword</code></td>
<td>optional</td>
<td>Password for the key referenced by <code>httpsAlias</code>. One of this or <code>httpsKeyPassword</code> is required.</td>
</tr>
</tbody>
</table>

Keystores can have a password and a separate (and, optionally, different) password for each key in the store. If you use the same password for both the keystore and the key at the given alias, you
only need to set one of httpsKeyStorePassword and httpsKeyPassword in your configuration. If only one of the two properties is set, the value of that property will be used for both properties. If both are set, each one will be used as configured.

In the follow example configuration, the password password would be used to decrypt both the keystore and the private key at alias example:

```json
{
    "httpPort": 8080,
    "httpsPort": 8443,
    "listenAddress": "0.0.0.0",
    "httpsKeyStoreFile": "/path/to/example.com.keystore",
    "httpsAlias": "example",
    "httpsKeyStorePassword": "password"
}
```

### 5.14 Migration across Cordra Environments

There are three scenarios when there may be a need to migrate from one Cordra environment to another:

- Routine upgrade of Cordra software.
- Migrating from Cordra v1 software to Cordra v2.
- Changing deployment infrastructure.

The migration steps needed in each case are discussed below.

**Warning:** The migration tools included within the Cordra software are designed not to lose data. However, as a precaution, it is a good idea to back up your existing data before proceeding with migration.

#### 5.14.1 Routine Upgrade

If you are using a particular version of the Cordra software, such as v2.x.y, and noticed a new patch or minor release (i.e., when x and y in the version number has changed), then it should be straightforward to upgrade the Cordra software. Depending on whether you are using Cordra as a standalone process or running Cordra as a war file within a servlet container, the steps will be slightly different.

Please pay particular attention to the release notes to notice if any specific migration steps are needed, and follow those instructions and adjust the instructions that are stated below accordingly:

- You should stop the existing Cordra process (or processes in case of a distributed Cordra system).
- In the case of Cordra war file being run inside a servlet container, replace that war file with the latest war file from the new release.
• In the case of Cordra being run as a standalone process, proceed to do a regular installation with the new release and just at the point when the Cordra process should be started, copy over the data directory from the previous installation into the new Cordra deployment directory. Also delete the repoInit.json file from the new Cordra environment.

• You may now proceed to start the Cordra process as before.

5.14.2 Migrating from Cordra v1 to Cordra v2

If you are using Cordra v1 software, we recommend that you take time to migrate to Cordra v2 software. The steps to do so are as follows:

• Make a backup copy of the Cordra v1 data directory.
• Download and unzip Cordra v2.
• Delete the default Cordra v2 data directory.
• Copy the Cordra v1 data directory into Cordra v2 installation folder.
• Start Cordra v2.

On startup, the Cordra v2 software will notice that the information in the data directory is from Cordra v1 and will migrate all data and configuration as needed. Files that are no longer needed will be moved into a directory named data/migration_delete_me. Once you have confirmed that the migration was successful, you can delete this folder.

After upgrading to Cordra v2, you may wish to move to a distributed system. To do so, simply follow the steps outlined in the next section.

5.14.3 Changing Deployment Infrastructure

There may be cases when you intend to change the deployment infrastructure. This could be because you intend to move from a single Cordra instance setup to multiple Cordra instance setup or vice versa. Or this could be because you intend to move from one storage technology (say MongoDB) to another (say Amazon S3) and/or from one indexing technology (say Apache Solr) to another (say Elasticsearch).

5.14.3.1 Changes to Indexing backend

If your change in deployment infrastructure does not involve a change to storage backend, but rather a change to just the indexing backend, then follow the steps outlined below:

• Setup the new indexing backend.
• Stop the Cordra process (or processes in case of a distributed Cordra system).
• Update the Cordra config.json to point at the new indexing backend coordinates. For details, refer to the Configuring Indexing Backend document.
• Start the Cordra process or processes.
• Issue reindexing. For details, refer to the Reindexing document.

5.14.3.2 Changes to Storage backend

If your change in deployment infrastructure affects the storage backend, we recommend you to export all the digital objects from your existing Cordra environment and import them into the new environment. For this,

• Ensure the new environment is ready for use.
• In a standalone Cordra scenario, delete the repoInit.json file from the new Cordra environment.
• In the case of a distributed Cordra system, do not create a znode within ZooKeeper for /cordra/repoInit.json.
• Stop the Cordra process (or processes in case of a distributed Cordra system)
• Export the digital objects using the import export tool from the old environment. For details, refer to the Import and Export Tools document.
• Import the output from the previous step into the new environment. For details, refer to the Import and Export Tools document.
• Start the Cordra process or processes.
• Issue reindexing. For details, refer to the Reindexing document.

5.14.4 Special Scenarios and Considerations

5.14.4.1 Change Prefix used with Identifiers

If you prefer to change the prefix of the identifiers associated with the digital objects, you should export digital objects, delete the digital objects from Cordra, make modifications on exported files, and import them back into Cordra.

The export tool exports digital objects into a directory called objects. Each file in that directory corresponds to a single digital object. Except for the payloads managed within the digital objects, the rest of the information is serialized as JSON that is conducive for editing using text search-and-replace tools such as sed.

Perform a global search for the current prefix and replace them with a new prefix. An example sed command that worked for us to replace prefix 20.5000.123 with 20.5000.456 is this:

```
sed -i 's/20\.5000\.123/20\.5000\.456/g' *
```

Run the above command from within the objects directory. Note that payloads are Base64 encoded in each of the exported file. If there are references to identifiers in those payloads, those identifiers will remain the same as before.
5.14.4.2 Other Artifacts

If you have extended Cordra and loaded additional Java libraries or configuration files into a Cordra environment, you should remember to specifically copy them into the new environment.
6.1 Calling Java from JavaScript

When writing type methods of life cycle hooks in JavaScript it is sometimes convenient to call Java code. Perhaps you have existing code or a 3rd party library that is only available in Java or perhaps you want to start a long running background process on another thread. In such cases it is possible to add a jar file to Cordra and call code in that jar file from JavaScript.

Any jar files you wish to call should be placed into a directory called lib in your Cordra data directory. If the lib directory doesn’t exist, create the lib directory in the Cordra data directory.

Consider the following simple Java class:

```java
package net.example;

public class Point {

    public double x;
    public double y;
    public double z;

    public Point(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    @Override
    public String toString() {
        return "(" + x + ", " + y + ", " + z + ")";
    }
}
```

To create an instance of a Point and call its toString method from a type method in JavaScript you would do the following:
exports.staticMethods = {}; exports.staticMethods.exampleStaticMethod = exampleStaticMethod;

function exampleStaticMethod(context) {
    const Point = Java.type("net.example.Point");
    const point = new Point(1.3, 3.44, 2.58);
    const result = {
        point: point.toString()
    };
    return result;
}


### 6.1.1 CordraHooksSupport

If you find yourself with a need to execute a long running Java background process you may want to listen for a shutdown event when Cordra shuts down to cleanly terminate your code. Support for this is provided by the class `CordraHooksSupport`. The below show example code that uses `CordraHooksSupport` to listen to the Cordra shutdown event. It also uses `CordraHooksSupport` to get an instance of a CordraClient that talks directly to the local Cordra instance.

```java
package net.example.background;

import net.cnri.cordra.CordraHooksSupport;
import net.cnri.cordra.CordraHooksSupportProvider;
import net.cnri.cordra.api.*;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;

public class BackgroundTasks {
    private static BackgroundTasks instance = null;
    private final ExecutorService exec;
    private final CordraClient cordra;
    private static volatile boolean shutdown = false;

    private BackgroundTasks() {
        exec = Executors.newSingleThreadExecutor();
        CordraHooksSupport hooks = CordraHooksSupportProvider.get();
        cordra = hooks.getCordraClient();
        hooks.addShutdownHook(this::shutdown);
    }
}
```

(continues on next page)
public synchronized static BackgroundTasks instance() {
    if (instance == null) {
        instance = new BackgroundTasks();
    }
    return instance;
}

public synchronized void shutdown() {
    if (shutdown) {
        return;
    }
    shutdown = true;
    exec.shutdown();
    try {
        exec.awaitTermination(Long.MAX_VALUE, TimeUnit.DAYS);
    } catch (InterruptedException e) {
    }
}

public synchronized void doBackgroundTask(String objectId) throws CordraException {
    exec.submit(() -> {
        try {
            CordraObject obj = cordra.get(objectId);
            //...
            //Do some long running task with obj
            //...
        } catch (Exception e) {
            
        }
    });
}

And then from a JavaScript instance method you could start the above Java background process:

exports.methods = {};
exports.methods.exampleInstanceMethod = exampleInstanceMethod;

function exampleInstanceMethod(obj, context) {
    const BackgroundTasks = Java.type("net.example.background.BackgroundTasks");
    const backgroundTasks = BackgroundTasks.instance();
    backgroundTasks.doBackgroundTask(obj.id);
}
6.2 Document Repository

Cordra can be configured to behave like a document repository. In fact, the Cordra distribution is configured this way by default, and includes definitions for three types of digital objects:

1. Document
2. User
3. Group

You can visit https://localhost:8443/ in a web browser to access the Cordra UI. Once you login as admin, you can create digital objects that represent users and groups.

Users can then login to create Document objects, i.e., digital objects of type Document. Cordra will index the metadata as well as payloads. Queries can then be issued and documents can be retrieved. Document objects can be shared with other users; likewise, type of access can be restricted to read or write operations. Versions can be created.

All of these UI features can be accessed via the APIs. The Medical Records Application goes into more details of how to configure Cordra using schemas, business rules, and access controls. That section also includes examples on how to access digital objects using the Cordra API.

6.3 Medical Records Application

We show below how to configure Cordra to behave like a sample Medical Records Application that could act as a starting point to create a comprehensive system for managing patient medical records.

6.3.1 Narrative

This sample application supports the following narrative:

- Patients visit Providers. Information is created during those visits, i.e., Encounters.
- Providers have areas of expertise that determine their speciality: General Physician and Endocrinologist are supported here.
- Patients encounters begin with a General Physician (GP) and the GP may order tests at the Labs.
- Lab tests produce additional information.
- GP may recommend patients to specialists.
- Patients encounter with specialists produces more information.
Also,

- Patients have read access to their medical information.
- Providers who see patients have read/write access to the medical information they create.
- In one possible scenario, patients must specifically give providers' access to their information when they visit new providers.
- In another possible scenario, the referring provider may give the referred provider access to the patient information.
- Labs do not have access to any medical information, by default. Patients must give explicit access.

We describe below how to support the aforementioned narrative, beginning with the design.

6.3.2 Design

We will create four different types of digital objects:

1. Patient type to capture information about patients. Patient Id, Name, Age, and Sex are captured.
2. Provider type to capture information about providers. Provider Id, Name, and Speciality are captured.
3. Lab type to capture information about labs. Lab Id and Name are captured.
4. Encounter type to capture medical information created by providers or generated due to lab tests. Specifically:
   - Timestamp,
   - Patient Id,
   - Producer (to reflect the Id of the provider and/or the lab that produced the information)
   - Notes (to capture provider observations or lab results)
   - Referred (to capture the Id of the provider or the lab to whom the patient is referred)
   - Referral Details (to capture lab prescriptions or details why the patient is referred to a specialist).

6.3.2.1 Authentication and Authorization

While we assume PKI will normally be enabled for a production application for security, and the system can require that patients, providers, and labs authenticate via private keys, we described the use of passwords below to keep the example simple. In addition to the aforementioned attributes for Patient, Provider, and Lab, we add password attribute as can be seen in the schemas below. In that sense, patients, providers, and labs are all treated as users of the system.
A patient has read access to encounters that are about them. A provider has read/write access to encounters that are generated by them. Other providers (or labs) will only have access to encounters if they are explicitly shared. We will see how later in this tutorial.

When signed in as the admin user select Admin->Authorization from the menu bar to specify type-level access control. This JSON is stored on the Design object and will be loaded automatically when you “Load from file” as described later on. For keeping this description simple, this particular authorization configuration allows Patients, Providers, and Labs to be read by all users but only be modified by the representative party. Encounter objects do not allow readers or writers at the type level as their particular readers and writers are calculated in JavaScript for each specific instance of an Encounter.

**Type-level Access Control:**

```
{
    "schemaAcls": {
        "CordraDesign": {
            "defaultAclRead": ["public"],
            "defaultAclWrite": [],
            "aclCreate": []
        },
        "Schema": {
            "defaultAclRead": ["public"],
            "defaultAclWrite": [],
            "aclCreate": []
        },
        "Patient": {
            "defaultAclRead": ["authenticated"],
            "defaultAclWrite": ["self"],
            "aclCreate": []
        },
        "Encounter": {
            "defaultAclRead": [],
            "defaultAclWrite": [],
            "aclCreate": ["authenticated"],
        },
        "Lab": {
            "defaultAclRead": [
```

(continues on next page)
6.3.2 Schemas

Four types of digital objects are show below: Patient, Provider, Lab, and Encounter. Notice that identifiers of digital objects are flagged to be auto-generated. The timestamp attribute in the Encounter object is also flagged to be auto-populated at the time of creation.

Patient Schema:

```json
{
   "type": "object",
   "properties": {
      "id": {
         "type": "string",
         "cordra": {
            "type": "autoGeneratedField",
            "handle": "id"
         }
      }
   }
}
```
"name": {
  "type": "string",
  "cordra": {
    "preview": {
      "showInPreview": true,
      "excludeTitle": true,
      "isPrimary": true
    }
  }
},
"age": {
  "type": "number",
  "cordra": {
    "preview": {
      "showInPreview": true
    }
  }
},
"sex": {
  "type": "string",
  "enum": [
    "male",
    "female",
    "other"
  ],
  "cordra": {
    "preview": {
      "showInPreview": true
    }
  }
},
"shareEncountersWith": {
  "type": "array",
  "format": "table",
  "uniqueItems": true,
  "items": {
    "type": "string",
    "cordra": {
      "type": {
        "handleReference": {
          "types": [
            "Provider",
            "Lab",
            "Lab"
          ]
        }
      }
    }
  }
}
Provider Schema:

```json
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "cordra": {
        "type": {
          "autoGeneratedField": "handle"
        }
      }
    },
    "name": {
      "type": "string",
      "cordra": {
        "preview": {
          "showInPreview": true,
          "isPrimary": true
        }
      }
    }
  }
}
```

(continues on next page)
"speciality": {
    "type": "string",
    "enum": [
        "General Physician",
        "Endocrinologist"
    ],
    "cordra": {
        "preview": {
            "showInPreview": true
        }
    }
},
"username": {
    "type": "string",
    "cordra": {
        "preview": {
            "showInPreview": true
        },
        "auth": "username"
    }
},
"password": {
    "type": "string",
    "format": "password",
    "cordra": {
        "auth": "password"
    }
}
}

Lab Schema:

{
    "type": "object",
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "autoGeneratedField": "handle"
            }
        }
    }
}
Encounter Schema:

```json
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string",
      "cordra": {
        "type": {
          "autoGeneratedField": "handle"
        }
      }
    },
    "timestamp": {
      "type": "string",
      "cordra": {
        "type": {
          "autoGeneratedField": "creationDate"
        }
      }
    }
  }
}
```
"notes": {  
  "type": "string",  
  "format": "textarea"
}  
},  
"shareWith": {  
  "type": "array",  
  "format": "table",  
  "uniqueItems": true,  
  "items": {  
    "type": "string",  
    "cordra": {  
      "type": {  
        "handleReference": {  
          "types": [  
            "Provider",  
            "Lab",  
            "Patient"
          ]
        }
      }
    }
  }
},  
"producer": {  
  "type": "string",  
  "cordra": {  
    "type": {  
      "autoGeneratedField": "createdBy"
    }
  }
},  
"patientId": {  
  "type": "string",  
  "cordra": {  
    "type": {  
      "handleReference": {  
        "types": [  
          "Patient"
        ]
      }
    }
  }
} 
} 
}}
6.3.2.3 Rules

Using JavaScript rules, we will ensure that an encounter can be read by the referenced patient. We will also ensure that an encounter can be read or written by the referenced provider or lab.

Associate the following JavaScript with the encounter type.

```javascript
var cordra = require("cordra");

exports.beforeSchemaValidation = beforeSchemaValidation;
exports.beforeDelete = beforeDelete;

function beforeDelete(encounter, context) {
  if (encounter.content.patientId === context.userId) {
    throw "Patients are not permitted to delete encounters";
  }
}

function beforeSchemaValidation(encounter, context) {
  if (encounter.content.patientId === context.userId) {
    authorizePatientPermittedToMakeRequest(encounter, context);
  }
}

encounter.acl = {};
```
encounter.acl.writers = [];
encounter.acl.readers = [];

addIfAbsent(encounter.acl.writers, encounter.content.patientId);
// Note that JavaScript prevents the patient from actually
// editing the content of the object, they can only edit shareWith property
addIfAbsent(encounter.acl.readers, encounter.content.patientId);

var producer = encounter.content.producer;
if (context.isNew) {
    producer = context.userId;
}
addIfAbsent(encounter.acl.writers, producer);
addIfAbsent(encounter.acl.readers, producer);

if (encounter.content.referred) {
    addIfAbsent(encounter.acl.readers, encounter.content.referred);
}
if (encounter.content.shareWith) {
    addAll(encounter.acl.readers, encounter.content.shareWith);
}

var patient = cordra.get(encounter.content.patientId);
if (patient.content.shareEncountersWith) {
    addAll(encounter.acl.readers, patient.content.shareEncountersWith);
}

return encounter;

function authorizePatientPermittedToMakeRequest(encounter, context) {
    if (context.isNew) {
        throw "A patient is not permitted to create encounters";
    }
    var oldEncounter = cordra.get(encounter.id);
    if (!isEqual(oldEncounter.acl.readers, encounter.acl.readers)) {
        throw "A patient is not permitted to directly modify the readers acl of an encounter";
    }
    if (!isEqual(oldEncounter.acl.writers, encounter.acl.writers)) {
        throw "A patient is not permitted to modify the writers acl of an encounter";
    }
    if (!isEqualWithoutShareWith(oldEncounter.content, encounter.content)) {
        throw "A patient is only permitted to modify the 'shareWith' property of an encounter";
    }
}
function isEqualWithoutShareWith(object, oldObject) {
    var objectCopy = JSON.parse(JSON.stringify(object));
    var oldObjectCopy = JSON.parse(JSON.stringify(oldObject));
    delete objectCopy.shareWith;
    delete oldObjectCopy.shareWith;
    return isEqual(objectCopy, oldObjectCopy);
}

function isEqual(a, b) {
    var aJson = JSON.stringify(a);
    var bJson = JSON.stringify(b);
    return aJson === bJson;
}

function addAll(list, idsToAdd) {
    for (var i = 0; i < idsToAdd.length; i++) {
        addIfAbsent(list, idsToAdd[i]);
    }
}

function addIfAbsent(list, id) {
    if (list.indexOf(id) == -1) {
        list.push(id);
    }
}

6.3.3 Setup

Download the above types here. You can then load this information using the Cordra UI. Sign in into Cordra as admin and select the Admin->Types dropdown menu. Click the “Load from file” button. In the dialog that pops up, select the types file you downloaded and check the box to delete existing objects. Click “Load” to import the types into Cordra.

That is it. The system is now ready for use.
6.3.4 Using the Application

We will use curl and the REST API to demonstrate how to use the system. JSON records used with the various commands are also shown below. Although not shown here, the Cordra UI may be used to perform equivalent actions.

For the purpose of this tutorial, the default Cordra address of https://localhost:8443/ is used. If your Cordra installation is running at a different location, please make the appropriate substitution. Also, the example curl commands will use the -k flag to tell curl to trust the self-signed certificate that comes with Cordra. This flag should not be used on production installations with real certificates.

All Ids shown in the sample curl commands were randomly generated. You will need to substitute these values with the appropriate Ids in your local system.

6.3.4.1 Authenticating

Before issuing any calls that require authorization, we must first authenticate and get an access token:

```
curl -k -X POST 'https://localhost:8443/auth/token' -H "Content-Type: application/json" --data @- << END
{
   "grant_type": "password",
   "username": "admin",
   "password": "password"
}
END
```

This call will return a token that we will use in subsequent calls.

6.3.4.2 Create users

Creations return back responses that consist of the Ids allotted to the corresponding user objects.

Create a provider who is a general physician:

```
{
   "id": "",
   "name": "Springfield Medical Centre",
   "speciality": "General Physician",
   "username": "gp",
   "password": "password"
}
END
```
Create a provider who is an endocrinologist:

```bash
{
  "id": "",
  "name": "Springfield Endocrinology",
  "speciality": "Endocrinologist",
  "username": "end",
  "password": "password"
}
END
```

Create a lab:

```bash
{
  "id": "",
  "name": "Generic Lab",
  "username": "lab",
  "password": "password"
}
END
```

Create a patient named Jane Smith:

```bash
{
  "id": "",
  "name": "Jane Smith",
  "age": 40,
  "sex": "female",
  "username": "jane",
  "password": "password"
}
END
```

Create another patient named John Smith. This patient is configured by the admin to share read access to all his new encounters with his wife Jane. As stated earlier, Jane’s Id as used in shareEncountersWith property is randomly generated in this example:

```bash
{
  "id": "",
  "name": "John Smith",
  "username": "john",
  "password": "password"
}
```

(continues on next page)
6.3.4.3 Create encounters

Note: For each different user in this section, you will need to get a new access token, as described above:

The patient visits the General Physician (GP). The GP creates this encounter. Note that id, producer, and timestamp are automatically filled by Cordra as instructed in schemas and in rules:

```
{
  "id": 
  "producer": 
  "timestamp": 
  "patientId": "test/e61a3587b3f7a142b8c7",
  "notes": "Patient complains of fatigue. Order CBC, BMP, and Thyroid Panel tests."
}
END
```

The GP orders lab tests. The patient visits the lab and the lab needs access to the encounter. The patient gives the lab access to the encounter. Since the patient can only modify the shareWith property on an encounter, first the patient, John, must GET the encounter object by its Id. As shown later, the patient can search for all encounters to get their Ids among other details:

```
```

Response:

```
{
  "id": "test/b4ab731228572b88fae1",
  "timestamp": "2018-09-19T19:48:52.430Z",
  "producer": "test/185108997731deb1edda",
  "patientId": "test/e61a3587b3f7a142b8c7",
  "notes": "Patient complains of fatigue. Order CBC, BMP, and Thyroid Panel tests."
}
```
The patient can then modify the encounter object, as received above, to include the Id of the lab in order to share the encounter. The patient updates the encounter object:

```
  ["test/9b405c77d1a2f17602f87"
END
```

The lab creates an encounter including the results from the lab. They include the GP in the shareWith property:

```
  {
    "id": "",
    "producer": "",
    "timestamp": "",
    "patientId": "test/e61a3587b3f7a142b8c9",
    "notes": "Thyroid Panel reveals high TSH levels.",
    "shareWith": [
      "test/185108997731deb1edda"
    ]
  }
END
```

The patient returns to the GP, and the GP creates a 3rd encounter referring the patient to an endocrinologist:

```
  {
    "id": "",
    "producer": "",
    "timestamp": "",
    "patientId": "test/e61a3587b3f7a142b8c9",
    "notes": "I am referring you to see an endocrinologist",
    "referred": "test/eb02409feb90de550756"
  }
END
```

The patient searches and retrieves all the encounters:

The patient modifies the previous two encounters sharing them with the endocrinologist:

[  
  "test/9b405c77d1a2f1760287",
  "test/e02409feb90de550756"
]
END

[  
  "test/185108997731deb1eddad",
  "test/e02409feb90de550756"
]
END

6.4 Person Registry

This example demonstrates how to configure Cordra to behave like a Registry of People. To showcase that such a registry could be used to manage transactions related to people, such as loans taken by them, we also show how loan digital objects could be managed and associated with person digital objects. This example is a basic starting point for a potential application.

In addition to showing how to configure Cordra, we will demonstrate how to interact with Cordra using the DOIP Java SDK.

6.4.1 Narrative

This sample application supports the following narrative:

- Person digital objects are added to the registry by administrators. The registry can track various biographical information, as well as information related to official documents.

- When creating or updating a person digital object, the request should fail if it would create a duplicate Person object. Duplicates are flagged if either the government issued ids match, or the last name and the date of birth match.

- Loan digital objects can be added, including information about its payment status as well as borrowers.
• There are operations for getting a list of all households, getting a list of all living persons with expired fingerprints, and getting a list of all living persons with defaulted loans.

• Information is not publicly accessible. User accounts can be created and added to one of two groups: an admin group that has read and write access to all records, and a users group that has read-only access.

6.4.2 Sample Objects

If you would like to follow along without having to manually create the objects described below, you can be download a file containing sample objects here. Once downloaded, you can then load this information into Cordra using the Cordra UI. Sign into Cordra as admin and select the Admin-Types dropdown menu. Click the “Load from file” button. In the dialog that pops up, select the file you downloaded and check the box to delete existing objects. Click “Load” to import the types into Cordra.

These sample objects are for testing only. They include users “admin1”, “admin2”, “user1”, and “user2”, all with the default password of “password”.

6.4.3 Design

6.4.3.1 Authentication and Authorization

We will need two groups for access controls. One of the groups will have read/write access to the objects, and the other group will only have read access.

Edit the Cordra Authorization configuration in the Design Object, and replace the existing json with the following:

```json
{
    "schemaAcls": {
        "User": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [ "self", "test/c5a2dc3dcba8c9c790" ],
            "aclCreate": [ "test/c5a2dc3dcba8c9c790" ]
        },
        "CordraDesign": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [],
            "aclCreate": []
        },
        "Schema": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [],
            "aclCreate": []
        },
        "Person": {
```
Because the default read permission is set to authenticated above, users will be required to log in before being able to read digital objects in the system. The default write and create permissions are set to the admin group, so non-admin users are limited to read-only access. Note that, if you are not using the sample data provided along with this tutorial, you will need to change test/c5a2dc3dcb24a8c9c790 to the id of the admin group you created.
6.4.3.2 Schemas

The schemas for the Person and Loan types of digital objects are show below. Notice that identifiers of digital objects are flagged to be auto-generated. The various timestamp attributes in the objects are also flagged to be auto-populated.

Person Schema:

```json
{
  "type": "object",
  "title": "Person",
  "required": [
    "id",
    "name",
    "birth",
    "gender",
    "address",
    "issuedIds"
  ],
  "properties": {
    "id": {
      "type": "string",
      "cordra": {
        "type": {
          "autoGeneratedField": "handle"
        }
      }
    }
  },
  "name": {
    "type": "object",
    "title": "Name",
    "required": [
      "first",
      "last"
    ],
    "properties": {
      "last": {
        "type": "string",
        "title": "Surname",
        "cordra": {
          "preview": {
            "showInPreview": true,
            "isPrimary": true
          }
        }
      },
      "first": {
```

(continues on next page)
"type": "string",
"title": "First Name",
"cordra": {
  "preview": {
    "showInPreview": true
  }
}
},
"middle": {
  "type": "string",
  "title": "Middle Name",
  "cordra": {
    "preview": {
      "showInPreview": true
    }
  }
}
},
"birth": {
  "type": "object",
  "title": "Birth Information",
  "required": [
    "date"
  ],
  "properties": {
    "date": {
      "type": "string",
      "pattern": "^[1-2][0-9][0-9][0-9][0-9][0-9]$",
      "title": "Date of Birth (YYYYMMDD)"
    },
    "certificate": {
      "type": "object",
      "title": "Birth Certificate",
      "required": [
        "id",
        "source"
      ],
      "properties": {
        "id": {
          "type": "string",
          "title": "Certificate ID"
        },
        "source": {
          "type": "string",
          "title": "Certificate Source"
        }
      }
    }
  }
}
}
"title": "Certificate Source"
"
"death": {
  "type": "object",
  "title": "Death Information",
  "required": [
    "date"
  ],
  "properties": {
    "date": {
      "type": "string",
      "pattern": "^[1-2][0-9][0-9][0-9][0-9]$",
      "title": "Date of Death (YYYYMMDD)"
    },
    "certificate": {
      "type": "object",
      "title": "Death Certificate",
      "required": [
        "id",
        "source"
      ],
      "properties": {
        "id": {
          "type": "string",
          "title": "Certificate ID"
        },
        "source": {
          "type": "string",
          "title": "Certificate Source"
        }
      }
    }
  }
},
"gender": {
  "type": "string",
  "title": "Gender",
  "enum": [
    "female",
    "male",
    "other"
  ]
}
"address": {
  "type": "object",
  "title": "Address",
  "required": [
    "line1",
    "line2"
  ],
  "properties": {
    "line1": {
      "type": "string",
      "title": "Line 1"
    },
    "line2": {
      "type": "string",
      "title": "Line 2"
    },
    "line3": {
      "type": "string",
      "title": "Line 3"
    }
  }
},
"issuedIds": {
  "type": "array",
  "title": "Government Issued Ids",
  "format": "table",
  "uniqueItems": true,
  "minItems": 1,
  "items": {
    "type": "object",
    "required": [
      "type",
      "id"
    ],
    "properties": {
      "id": {
        "type": "string",
        "title": "ID"
      },
      "type": {
        "type": "string",
        "title": "ID Type"
      }
    }
  }
}
"fingerprints": {
  "type": "object",
  "title": "Finger Print External Reference",
  "required": [
    "id",
    "source",
    "lastCapturedDate"
  ],
  "properties": {
    "id": {
      "type": "string",
      "title": "Certificate ID"
    },
    "source": {
      "type": "string",
      "title": "Certificate Source"
    },
    "lastCapturedDate": {
      "type": "string",
      "pattern": "^[1-2][0-9]{7}$",
      "title": "Last Captured Date (YYYYMMDD)"
    }
  }
},
"recordCreatedOn": {
  "type": "string",
  "title": "Record Creation Date",
  "cordra": {
    "type": {
      "autoGeneratedField": "creationDate"
    }
  }
},
"recordModifiedOn": {
  "type": "string",
  "title": "Record Modification Date",
  "cordra": {
    "type": {
      "autoGeneratedField": "modificationDate"
    }
  }
},

(continues on next page)
Loan Schema:

```json
{
    "type": "object",
    "title": "Loan",
    "required": ["id", "status", "lender", "start", "end", "borrowers", "amount"],
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "type": {
                    "autoGeneratedField": "handle"
                }
            }
        },
        "lender": {
            "type": "string",
            "cordra": {
                "type": {
                    "autoGeneratedField": "modifedBy"
                }
            }
        }
    }
}
```
"title": "Lender"
},
"start": {
  "type": "string",
  "pattern": "^[1-2]{1}[0-9]{7}$",
  "title": "Term Start Date (YYYYMMDD)"
},
"end": {
  "type": "string",
  "pattern": "^[1-2]{1}[0-9]{7}$",
  "title": "Term End Date (YYYYMMDD)"
},
"status": {
  "type": "string",
  "title": "Status",
  "enum": [
    "defaulted",
    "performing",
    "paid"
  ]
},
"amount": {
  "type": "object",
  "title": "Loan Amount",
  "required": [
    "amount",
    "currency"
  ],
  "properties": {
    "amount": {
      "type": "number",
      "minimum": 0,
      "title": "Amount"
    },
    "currency": {
      "type": "string",
      "title": "Currency"
    }
  }
},
"borrowers": {
  "type": "array",
  "title": "Borrowers",
  "uniqueItems": true,
  "minItems": 1,
"items": {  
  "type": "string",  
  "title": "Borrower",  
  "cordra": {  
    "type": {  
      "handleReference": {  
        "types": [  
          "Person"  
        ]  
      }  
    }  
  }  
},  
"recordCreatedOn": {  
  "type": "string",  
  "title": "Record Creation Date",  
  "cordra": {  
    "type": {  
      "autoGeneratedField": "creationDate"  
    }  
  }  
},  
"recordModifiedOn": {  
  "type": "string",  
  "title": "Record Modification Date",  
  "cordra": {  
    "type": {  
      "autoGeneratedField": "modificationDate"  
    }  
  }  
},  
"recordCreatedBy": {  
  "type": "string",  
  "title": "Record Created By",  
  "cordra": {  
    "type": {  
      "autoGeneratedField": "createdBy"  
    }  
  }  
},  
"recordModifiedBy": {  
  "type": "string",  
  "title": "Record Modified By",  
  "cordra": {  
    "type": {  
      "autoGeneratedField": "modifiedBy"  
    }  
  }  
}
6.4.3.3 Rules and Type Methods

We will use the beforeSchemaValidation lifecycle hook to de-duplicate persons on create and update. Additionally, we will add two static type methods to the Person type for getting households and persons with expired fingerprints.

Person Javascript:

```javascript
const cordra = require('cordra');
let prefix = cordra.get("design").content.handleMintingConfig.prefix;
if (!prefix) prefix = 'test';

exports.beforeSchemaValidation = beforeSchemaValidation;
exports.staticMethods = {};
exports.staticMethods[prefix+"/Op.GetExpiredFingerPrints"] = getExpiredFingerPrints;

function beforeSchemaValidation(obj, context) {
    // dedupe and rejects creations or updates based on the following criteria:
    // - If the government issued ids match.
    // - If the last name and the date of birth match
    let query = '+type:Person';
    query += `+(+/name/last:"${obj.content.name.last}" +/birth/date:"${obj.content.birth.date}");
    obj.content.issuedIds.forEach(id => {
        query += ` OR (+/issuedIds/_/id:"${id.id}" +/issuedIds/_/type:"${id.type}");
    });
    query += ');
    const res = cordra.search(query);
    if (res.size > 0) {
        throw "Either Issued Id or Last Name + Birthday is duplicated.";
    }
    return obj;
}

function getHouseholdMembers(context) {

(continues on next page)
// Returns groups of Person Ids consisting of people who are still alive and from the same address.
const households = {};
const livingPersons = cordra.search('+type:"Person" ~/death/date:[* TO */]');
livingPersons.results.forEach(person => {
  const addressHash = getAddressHash(person.content.address);
  if (addressHash) {
    if (!households.hasOwnProperty(addressHash)) {
      households[addressHash] = [];
    }
    households[addressHash].push(person.id);
  }
});
return Object.keys(households).map(i => households[i]);

function getAddressHash(address) {
  if (!address) return null;
  const lines = address.line1 + address.line2 + address.line3;
  return lines.hashCode();
}

function getExpiredFingerPrints(context) {
  // Lists all Ids of living people that have lastCapturedDate before the given expirationDate.
  let expirationDate = context.params.expirationDate;
  if (!expirationDate) expirationDate = '*';
  const persons = cordra.search('+type:"Person" ~/death/date:[* TO */] +/fingerprints/lastCapturedDate:[* TO ' + expirationDate + ']');
  return persons.results.map(p => p.id);
}

Note: The de-duplication function shown is best-effort using search and only checking a few properties. A production system would need to be more thorough (to disallow any simultaneous create/update requests that might be potential duplicates), but this is sufficient for demonstration purposes.

We also need to add a static type method to the Loan type, for getting a list of persons with defaulted loans.

Loan Javascript:

const cordra = require('cordra');
let prefix = cordra.get("design").content.handleMintingConfig.prefix;
if (!prefix) prefix = 'test';
exports.staticMethods = {};

function getDefaulters(context) {
    const livingPersons = cordra.search('+type:"Person" -/death/date:[* TO *]');
    const personIds = livingPersons.results.map(p => p.id);
    const defaultedLoans = cordra.search('+type:"Loan" +/status:"defaulted"');
    const defaultedBorrowers = [];
    defaultedLoans.results.forEach(loan => {
        loan.content.borrowers.forEach(borrower => {
            if (personIds.indexOf(borrower) !== -1) {
                defaultedBorrowers.push(borrower);
            }
        });
    });
    return defaultedBorrowers.filter(b => personIds.indexOf(b) !== -1);
}

6.4.4 Using the Application

We will use the DOIP Java SDK to demonstrate how to use the system. Although not shown here, the Cordra UI or any other Cordra client library may be used to perform equivalent actions.

All identifiers shown are the randomly-generated identifiers included in the sample objects. If you have created your own objects instead of importing the sample ones, you will need to substitute these values with the appropriate identifiers in your local system.

6.4.4.1 Client Setup

Before proceeding, be sure you have configured your Java project with doip-sdk jar file on the classpath. You can find instructions for doing so here.

First, we must create a DOIP client Java instance, along with authentication and service information for the client to use.

```java
String serviceId = "test/service";
DoipClient client = new DoipClient();
ServiceInfo serviceInfo = new ServiceInfo(serviceId, "localhost", 9000);
AuthenticationInfo adminAuthInfo = new PasswordAuthenticationInfo("admin1", "password...");
```

Change the authentication and service information as appropriate to your system. You may prefer to use a PKI based setup.
6.4.4.2 Basic Operations

Create a Person:

```java
DigitalObject personDobj = new DigitalObject();
personDobj.type = "Person";
JsonObject personContent = new JsonObject();
JsonObject name = new JsonObject();
name.addProperty("first", "Jane");
name.addProperty("last", "Doe");
personContent.add("name", name);
personContent.addProperty("gender", "female");
JsonObject address = new JsonObject();
address.addProperty("line1", "123 Elm St");
address.addProperty("line2", "Apt A");
address.addProperty("line3", "Charlottesville, VA 22902");
personContent.add("address", address);
JsonObject birth = new JsonObject();
birth.addProperty("date", "19330202");
personContent.add("birth", birth);
JsonObject death = new JsonObject();
deadth.addProperty("date", "20200115");
personContent.add("death", death);
JsonObject issueIds = new JsonArray();
JsonObject id = new JsonObject();
id.addProperty("type", "license");
id.addProperty("id", "123456-789");
issueIds.add(id);
personContent.add("issuedIds", issueIds);
JsonObject prints = new JsonObject();
prints.addProperty("id", "qwerty123456");
prints.addProperty("source", "govt");
prints.addProperty("lastCapturedDate", "19750306");
personContent.add("fingerprints", prints);
personDobj.setAttribute("content", personContent);
personDobj = client.create(personDobj, adminAuthInfo, serviceInfo);
System.out.println("Created Person: "+ personDobj.id);
```

Create a Loan:

```java
DigitalObject loanDobj = new DigitalObject();
loanDobj.id = "test/a4a33fed1bba7e8752ea";
loanDobj.type = "Loan";
JsonObject loanContent = new JsonObject();
loanContent.addProperty("lender", "ABC Bank");
loanContent.addProperty("start", "20180101");
loanContent.addProperty("end", "20220101");
```

(continues on next page)
Cordra Technical Manual, Release 2.5.2

(continued from previous page)

```java
loanContent.addProperty("status", "performing");
JsonObject amount = new JsonObject();
amount.addProperty("amount", 10000);
amount.addProperty("currency", "USD");
loanContent.add("amount", amount);
JsonArray borrowers = new JsonArray();
borrowers.add(personDobj.id);
loanContent.add("borrowers", borrowers);
loanDobj.setAttribute("content", loanContent);
loanDobj = client.create(loanDobj, adminAuthInfo, serviceInfo);
System.out.println("Created Loan: " + loanDobj.id);

Retrieve:

loanDobj = client.retrieve(loanDobj.id, adminAuthInfo, serviceInfo);

Update an object:

loanDobj.attributes.get("content").getAsJsonObject().addProperty("lender", "XYZ Bank");
loanDobj = client.update(loanDobj, adminAuthInfo, serviceInfo);

Delete: See below, as these code blocks are organized such that they can be transferred in order into a Java method.

Search:

QueryParams queryParams = new QueryParams(0, 50);
String query = "*:*";
try (SearchResults<DigitalObject> results = client.search(serviceId, query, queryParams, adminAuthInfo, serviceInfo)) {
    System.out.println("Search: Found " + results.size() + " objects:");
    for (DigitalObject result : results) {
        System.out.println("\t" + result.id + ": " + result.type);
    }
}

List Operations:

List<String> ops = client.listOperations("test/8c41ae88467fe5bbad09", adminAuthInfo, serviceInfo);
System.out.println("Operations available on Person schema:");
ops.forEach(System.out::println);
ops = client.listOperations("test/36c71e41f2d1e438ced9", adminAuthInfo, serviceInfo);
System.out.println("Operations available on Loan schema:");
ops.forEach(System.out::println);
```
6.4.4.3 Extended Operations

Op.GetExpiredFingerPrints:

```java
JsonObject params = new JsonObject();
params.addProperty("expirationDate", "20200101");
try (DoipClientResponse resp = client.performOperation("test/8c41ae88467fe5bbad09", "test/Op.GetExpiredFingerPrints", adminAuthInfo, null, params, serviceInfo)) {
    if (resp.getStatus().equals(DoipConstants.STATUS_OK)) {
        try (InDoipMessage in = resp.getOutput()) {
            InDoipSegment firstSegment = InDoipMessageUtil.getFirstSegment(in);
            if (firstSegment == null) {
                throw new DoipException("Missing first segment in response");
            }
            Gson gson = new GsonBuilder().setPrettyPrinting().create();
            System.out.println(gson.toJson(firstSegment.getJson()));
        }
    } else {
        throw DoipClient.doipExceptionFromDoipResponse(resp);
    }
}
```

Op.GetHouseholdMembers:

```java
try (DoipClientResponse resp = client.performOperation("test/8c41ae88467fe5bbad09", "test/Op.GetHouseholdMembers", adminAuthInfo, null, serviceInfo)) {
    if (resp.getStatus().equals(DoipConstants.STATUS_OK)) {
        try (InDoipMessage in = resp.getOutput()) {
            InDoipSegment firstSegment = InDoipMessageUtil.getFirstSegment(in);
            if (firstSegment == null) {
                throw new DoipException("Missing first segment in response");
            }
            Gson gson = new GsonBuilder().setPrettyPrinting().create();
            System.out.println(gson.toJson(firstSegment.getJson()));
        }
    } else {
        throw DoipClient.doipExceptionFromDoipResponse(resp);
    }
}
```

Op.GetDefaulters:

```java
try (DoipClientResponse resp = client.performOperation("test/36c71e41f2d1e438ced9", "test/Op.GetDefaulters", adminAuthInfo, null, serviceInfo)) {
    if (resp.getStatus().equals(DoipConstants.STATUS_OK)) {
        try (InDoipMessage in = resp.getOutput()) {
```

(continues on next page)
InDoipSegment firstSegment = InDoipMessageUtil.getFirstSegment(in);
if (firstSegment == null) {
    throw new DoipException("Missing first segment in response");
}

Gson gson = new GsonBuilder().setPrettyPrinting().create();
System.out.println(gson.toJson(firstSegment.getJson()));
else {
    throw DoipClient.doipExceptionFromDoipResponse(resp);
}

6.4.4.4 Payloads

Binary files can be attached to Person or Loan objects as payloads. Payloads can be files such as pictures, scans of certificates or government issued ids, or documents.

For example, to add a payload to an existing Loan object:

```java
DigitalObject loanDobj = client.retrieve("test/a4a33fed1bba7e8752ea", adminAuthInfo, serviceInfo);
Element payload = new Element();
payload.type = "application/pdf";
payload.id = "contract.pdf";
payload.in = Files.newInputStream(Paths.get("contract.pdf")); // Change this to a file on your local system
payload.attributes = new JsonObject();
payload.attributes.addProperty("filename", "contract.pdf");
loanDobj.elements = new ArrayList<>();
loanDobj.elements.add(payload);
client.update(loanDobj, adminAuthInfo, serviceInfo);
```

6.4.4.5 Deduplication and Access Controls

If you try to create a duplicate person, the beforeSchemaValidation lifecycle hook will block the change. For example, running this code (which duplicates one of the sample persons):

```java
DigitalObject personDobj = new DigitalObject();
personDobj.type = "Person";
JsonObject personContent = new JsonObject();
JsonObject name = new JsonObject();
name.addProperty("first", "Jane");
name.addProperty("last", "Hoeger");
```
personContent.add("name", name);
personContent.addProperty("gender", "female");
JsonObject address = new JsonObject();
address.addProperty("line1", "123 Elm St");
address.addProperty("line2", "Apt A");
address.addProperty("line3", "Charlottesville, VA 22902");
personContent.add("address", address);
JsonObject birth = new JsonObject();
birth.addProperty("date", "19941221");
personContent.add("birth", birth);
JsonObject id = new JsonObject();
id.addProperty("type", "license");
id.addProperty("id", "123456-789");
issueIds.add(id);
personContent.add("issuedIds", issueIds);
JsonObject prints = new JsonObject();
prints.addProperty("id", "qwerty123456");
prints.addProperty("source", "govt");
prints.addProperty("lastCapturedDate", "19750306");
personContent.add("fingerprints", prints);
personDobj.setAttribute("content", personContent);
client.create(personDobj, adminAuthInfo, serviceInfo);

Will produce an error like this:

```
Exception in thread "main" net.dona.doip.client.DoipException: Bad request: 400;...
  → Either Issued Id or Last Name + Birthday is duplicated.
```

Trying to create objects as a user will also fail, due to the Authorization configuration. For example, this Loan creation attempt:

```
DigitalObject loanDobj = new DigitalObject();
loanDobj.type = "Loan";
JsonObject loanContent = new JsonObject();
loanContent.addProperty("lender", "ABC Bank");
loanContent.addProperty("start", "20180101");
loanContent.addProperty("end", "20220101");
loanContent.addProperty("status", "performing");
JsonObject amount = new JsonObject();
amount.addProperty("amount", "10000");
amount.addProperty("currency", "USD");
loanContent.add("amount", amount);
JsonArray borrowers = new JsonArray();
borrowers.add("test/987a25ef7a329664b150");
loanContent.add("borrowers", borrowers);
```
loanDobj.setAttribute("content", loanContent);
// Using user authentication info instead of admin
AuthenticationInfo userAuthInfo = new PasswordAuthenticationInfo("user1", "password");
client.create(loanDobj, userAuthInfo, serviceInfo);

Will produce an error like this:

Exception in thread "main" net.dona.doip.client.DoipException: Forbidden: 403

However, users can still read object and call read-only static methods. Unauthenticated users who try to read objects instead get a Unauthorized: 401 error.

6.4.4.6 Basic Operations (continued)

Delete objects:

client.delete(personDobj.id, adminAuthInfo, serviceInfo);
client.delete(loanDobj.id, adminAuthInfo, serviceInfo);

6.5 User Registration

Cordra software enables only authenticated users, including the admin, to be able to create digital objects. Since users are also represented as digital objects, it may be necessary to allow new users to initiate the creation of their digital objects, while disallowing the creation of any other digital objects.

This section describes how to build a user registration workflow within Cordra that enables new users to create their accounts in Cordra and subsequently activate their accounts. Instead of allowing any unauthenticated user to create an account in one step, we will bind a user account in Cordra to the user's email address.

In particular, when a request for the creation of a new account (i.e., user object) is received, Cordra generates a random string (called activation key) and sends an email to the requesting user’s email address. Only that user will then be in possession of both the activation key and the information supplied with the creation request. Cordra verifies that that is the case and completes the user account creation.

Specifically, the following features are considered for this user registration workflow:

- Allow new users to create an account along with their email address in Cordra by themselves. Only admins can create user accounts by default.
- Cordra to send emails with activation keys for users to confirm that they initiated the account creation.
• Allow users to activate their account without any admin intervention using the activation key sent to their email address. (This step implicitly also confirms that the users have access to emails corresponding to the registered email address).

We will start this description with a default Cordra distribution and highlight all of the changes necessary to add the desired features. Specifically, we will need to:

• Modify the authorization for the User schema so that unauthenticated users can create User objects.
• Add Type Methods in JavaScript to the User schema for sending emails and confirming accounts.
• Add the necessary support files to the Cordra data directory to enable sending emails through a Type method.

6.5.1 Steps

6.5.1.1 Get Cordra

You should download the default Cordra distribution here. Once you have downloaded the zip file, unzip it, and start Cordra as explained here. Once the startup process is complete, you should be able to access the Cordra UI at https://localhost:8443.

6.5.1.2 Enable Email Support

For testing purposes, you can skip enabling the email support. If you do not set up email support, be sure your JavaScript method prints the activation key to the terminal console where the Cordra process is launched, so that you can use it to activate the user account, for testing purposes. In production, however, email support should be enabled as described in Sending Emails.

6.5.1.3 Modify the Authorization Config

By default, unauthenticated users cannot create User objects in Cordra. To allow users to create their own accounts, log into the Cordra UI as the admin user, using the password you created on first startup. Choose the Authorization menu item from the Admin menu at the top of the page. You will be presented with a JSON representing the current default access controls. Replace the existing JSON with the following:

```
{
    "schemaAcls": {
        "User": {
            "defaultAclRead": [ "public" ],
            "defaultAclWrite": [ "self" ],
            "aclCreate": [ "public" ],
            "aclMethods": {
                "instance": {
                ...
```
By performing the above steps, you have added a new ACL for the User type (lines 3-15). Previously, there was not a separate ACL for the type, and so it used the default ACL, which says that only authenticated users can create objects of this type. The new ACL says that anyone can create User objects, and any user can modify their own object. Also, we have added public permission to run the `activateAccountIfKeyIsValid` instance method, which is described below.

Because users can modify their own object, you will need to be careful about what is stored in that object. For example, it is probably ok if users can deactivate their account, but if user objects can include properties that only administrators can view or those that empower any user into an administrator status, then do not allow users access to such properties.
6.5.1.4 Modify User Schema

You will have to modify the User type to add the functionality needed to support the registration process. Type methods will be used to generate and send the verification key, and confirm the key and activate the account.

Creating New User Activation Key

When a user creation request is received by Cordra, Cordra should flag the new account as inactive, generate an activation key for the account and store it securely, and email that key to the user. To configure Cordra to do that, you will need to modify the User schema to add the necessary fields for active status and the activation key. You should edit the schema and replace the existing JSON with the following:

```json
{
    "type": "object",
    "required": [
        "id",
        "username",
        "password",
        "email",
        "isActive",
    ],
    "properties": {
        "id": {
            "type": "string",
            "cordra": {
                "type": {
                    "autoGeneratedField": "handle"
                }
            }
        },
        "username": {
            "type": "string",
            "title": "Username",
            "cordra": {
                "preview": {
                    "showInPreview": true,
                    "isPrimary": true
                },
                "auth": "username"
            }
        },
        "password": {
            "type": "string",
            "format": "password",
```
By following the above steps, you have added the fields `email`, `isActive`, and `activationKey`, and made the `email` and `isActive` field required. Special flags are added using the `cordra` property to...
indicate that isActive and activationKey should be treated differently. The accountActive property (line 63) means that Cordra should use this field to indicate whether or not the user account is active. The secureProperty flag (line 71) means that this field will be hashed and salted before storage and will never be stored as plain text, so other existing users including the admin cannot view the key.

Next, you will have to modify the beforeSchemaValidation method on the User object to generate and save the key, as well as email it to the user. You should replace the default User javascript with the following:

```javascript
exports.beforeSchemaValidation = beforeSchemaValidation;

var emailConfig = {
  "fromAddress": "admin@example.com",
  "subject": "testing javascript email",
  "textTemplate": "Your activation key is \{KEY\}.",
  "htmlTemplate": "<html><body><h1>Your activation key is \{KEY\}.</h1></body></html>",
};

function beforeSchemaValidation(obj, context) {
  if (!obj.content.id) obj.content.id = "";
  if (!obj.content.password) obj.content.password = "";
  if (!obj.content.email) obj.content.email = "";
  if (isEmailConfigured() && !isValidEmail(obj.content.email)) {
    throw "Email is invalid."
  }
  if (context.isNew) {
    obj.content.isActive = false;
    obj.content.activationKey = generateRandomString();
    sendKeyEmail(obj.content.email, obj.content.activationKey);
  }
  return obj;
}

function generateRandomString() {
  return Math.random().toString(36).substr(2, 15);
}

function isValidEmail(email) {
  var re = /\S+@\S+\./\S+;  // Email regex
  return re.test(email);
}

function sendKeyEmail(email, activationKey) {
  if (isEmailConfigured()) {
    var textMessage = emailConfig.textTemplate.replace("\{KEY\}", activationKey);
    var htmlMessage = emailConfig.htmlTemplate.replace("\{KEY\}", activationKey);
    // Email sending function
  }
```
sendEmail(email, emailConfig.fromAddress, emailConfig.subject, textMessage, htmlMessage);
} else {
    print(email + ": " + activationKey);
}
}

function sendEmail(toAddress, fromAddress, subject, textMessage, htmlMessage) {
    // Java types
    var EmailBuilder = Java.type("org.simplejavamail.email.EmailBuilder");
    var MailerBuilder = Java.type("org.simplejavamail.mailer.MailerBuilder");
    var TransportStrategy = Java.type("org.simplejavamail.api.mailer.config.
        TransportStrategy");

    // Build email
    var serverConfig = getServerConfig();
    var email = EmailBuilder.startingBlank()
        .to(toAddress)
        .from(fromAddress)
        .withSubject(subject)
        .withHTMLText(htmlMessage)
        .withPlainText(textMessage)
        .buildEmail();

    var mailerBuilder = MailerBuilder
        .withSMTPServer(serverConfig.serverAddress, serverConfig.serverPort, ...
        serverConfig.username, serverConfig.password)
        .withSessionTimeout(10000);
    if (serverConfig.enableStartTls) {
        mailerBuilder = mailerBuilder.withTransportStrategy(TransportStrategy.SMTP_
            "TLS");
    } else if (serverConfig.enableStartTls) {
        mailerBuilder = mailerBuilder.withTransportStrategy(TransportStrategy.SMTPS);
    }
    var mailer = mailerBuilder.buildMailer();
    mailer.sendMail(email);
}

function getConfigFilePath() {
    var dataDir = java.lang.System.getProperty("cordra.data");
    var filePath = java.nio.file.Paths.get(dataDir).resolve("emailServerConfig.json");
    return filePath;
}
A few things to note here:

- Email configuration for the email to be sent are hardcoded into the JavaScript lines (3-8).
- We are using `context.isNew` to make sure we only create an activation key on new object creation (line 17).
- `beforeSchemaValidation` runs before the object is stored, so we can modify the key and active flag appropriately (lines 18-19).
- Key generation is using a pseudo-random function for the purposes of this tutorial (line 34). For actual use, you will want to use a more secure key generation method.
- Email validation in the JavaScript is purposefully minimal (lines 29-32). We will validate the address by sending an email to it.
- It is possible to access Java classes in the JavaScript. The `sendEmail` function is using classes from the jar files we included earlier (lines 476-48).
- Email server config is loaded from local file in the `loadServerConfig` function (lines 72-76), so the secrets are never network accessible through Cordra. Note that we are also using Java here to read the file.

### Confirming a Key

Next, you should add a schema instance method for confirming a key by editing the `User` javascript and adding the following to the top:

```javascript
var cordraUtil = require('cordraUtil');

exports.methods = {};
exports.methods.activateAccountIfKeyIsValid = activateAccountIfKeyIsValid;
```

You should then add the following JavaScript function to the bottom:

```javascript
function activateAccountIfKeyIsValid(object, context) {
    var activationKey = context.params.activationKey;
    if (!activationKey) return false;
```
var success = cordraUtil.verifySecret(object, "/activationKey", activationKey);
if (!success) {
    throw "Could not verify key."
}
object.content.isActive = true;
delete object.content.activationKey;
return true;
}

Again, a few things to note:

- We are importing the built-in cordraUtil javascript module, which gives access to the verifySecret function used in the activateAccountIfKeyIsValid function. You can read more about the cordraUtil JavaScript module here.

- Any instance methods we create will only be available if added to the export.methods objects.

- Any changes made to the object in an instance method are automatically saved. Here, we are setting isActive to true and removing the activationKey.

6.5.2 Testing It Out

You should now be able sign up for a user account in Cordra, get an activation key, and use that key to activate the account. Here are some example curl commands for making the appropriate calls.

Create a new user account:

```bash
curl -k -X POST 'https://localhost:8443/objects/?type=User' -H "Content-Type: application/json" --data @- << END
{
    "username": "testUser",
    "password": "testPassword",
    "email": "test@example.com"
}
END
```

Activate the user account with the key:

```bash
curl -k -X POST 'https://localhost:8443/call/?objectId=test/a94a8fe5ccb19ba61c4c&method=activateAccountIfKeyIsValid' -H "Content-Type: application/json" --data @- << END
{
    "activationKey": "XXXXXXXXX"
}
END
```

Note that the objectId in the activation URI is the id of the User object for this account, not the id of the User type object. If you are following along, you may need to modify the URI with the id of
the User object in your local Cordra instance.

### 6.5.3 Admin-created Accounts

By following the steps above, you have built a system for allowing users to create and activate their own accounts. However, in some systems, an administrator creates the accounts for the user, and then the user activates the account and chooses their password. With a few changes to the code described above, you can support this alternative workflow.

First, you will need to modify the Authorization config to restrict the ability to create User objects. You can do this by simply removing the `public` create ACL for the User type. When complete, your User type ACL should look like this:

```json
"User": {
    "defaultAclRead": [ "public" ],
    "defaultAclWrite": [ "self" ],
    "aclCreate": [],
    "aclMethods": {
        "instance": {
            "activateAccountIfKeyIsValid": [ "public" ]
        },
        "default": {
            "instance": []
        }
    }
}
```

Note that the `aclCreate` property is now an empty list. This means that only the admin user will be able to create new User objects.

Next, you will have to modify `beforeSchemaValidation` to set a temporary password on the newly created user account. You can do that by changing the method to look like this:

```javascript
function beforeSchemaValidation(obj, context) {
    if (!obj.content.id) obj.content.id = "";
    if (!obj.content.password) obj.content.password = "";
    if (!obj.content.email) obj.content.email = "";
    if (!isValidEmail(obj.content.email)) {
        throw "Email is invalid."
    }
    if (context.isNew) {
        obj.content.isActive = false;
        obj.content.activationKey = generateRandomString();
        obj.content.password = generateRandomString();
        sendKeyEmail(obj.content.email, obj.content.activationKey);
    }
    return obj;
}
```

User Registration 275
The only change above is that we are setting the password to a random string on line 11.

Finally, the `activateAccountIfKeyIsValid` needs to set the new user’s password if the activation key is valid. To do that you can modify the method to look like this:

```javascript
function activateAccountIfKeyIsValid(object, context) {
  var activationKey = context.params.activationKey;
  if (!activationKey) return false;
  var newPassword = context.params.password;
  if (!newPassword || newPassword.length < 8) {
    throw "Password missing or too short. Must be at least 8 characters."
  }
  var success = cordraUtil.verifySecret(object, "/activationKey", activationKey);
  if (!success) {
    throw "Could not verify key."
  }
  object.content.isActive = true;
  object.content.password = newPassword;
  delete object.content.activationKey;
}
```

There are a few important changes made above. First, we are checking for the new password in the method context and doing a small amount of validation (lines 4-7). Object changes within a Type method do not go through validation, so be sure to do any validation you need in the method. Once the key is verified, you can set the new password on the object (line 14).

With the above changes in place, you can now test the new account registration workflow.

Because only admin is allowed to create user objects, we must first authenticate to get an access token to use with our curl command. Use the password you created when starting your Cordra instance:

```bash
curl -k -X POST 'https://localhost:8443/auth/token' -H "Content-Type: application/json" --data @- << END
{
  "grant_type": "password",
  "username": "admin",
  "password": "password"
}
END
```

Admin creates a new user account:

```bash
{
  "username": "testUser",
  "email": "test@example.com"
}
```

(continues on next page)
User activates their account with the key and a new password:

```bash
curl -k -X POST 'https://localhost:8443/call/?objectId=test/a94a8fe5cbe19ba61c4c&method=activateAccountIfKeyIsValid' -H "Content-Type: application/json" --data @- < END
{
    "activationKey": "XXXXXXXXX",
    "password": "newPassword"
}
END
```

Again, be sure to change the `objectId` in the URI to match the id of the User object being activated.

### 6.5.4 Full example

Configurations and code that you will need to follow this description is included in the Cordra download, in the `extensions/user-registration` directory. This includes the full User type object and Cordra Authorization config. It also includes a sample web application you can use to test out this functionality. To install the application, create a directory in your Cordra data directory called `webapps` and then copy the `demo` directory into the `webapps` directory. The demo will now be available at https://localhost:8443/demo.

### 6.5.5 Additional Thoughts

In this tutorial, we have explored a number of topics, including Cordra Type methods, access controls for objects and methods, and how to use third-party Java libraries in Type methods. This application is just an example, though. There are a few additional things to think about while implementing a secure user registration in a live system. For example:

- Throttling email sending and account creation.
- Using CAPTCHAs, two-factor authentication, or other alternative account verification methods.
- Expiring activation keys after a certain time.

These topics are important, but are considered out of scope.
6.6 Digital Object Linking

Digital objects in Cordra can be linked together in a verifiable way by combining digital object identifiers and digital object hashes. For details about hashing digital objects, see Digital Object Hashing. Consider that we want to create a sequence of blocks (Block objects), each of which references one or more digital objects. Specifically, say, the following linked digital objects are desired:

```
test/second-block
  |   +-->test/othello
  |     +-->test/first-block
  |        |   +-->test/macbeth
  |        |     +-->test/hamlet
```

The following conceptual schema for a type called Block will be used to enable the above desire: Each block contains a property previousBlock that includes a reference to the previous block. Additionally each block contains the property pointers which is an array of references to other Cordra objects of any type from this block. In this context, a pointer or a reference is a JSON object that includes an id and a hash, both pertaining to the digital object being pointed at.

You can download the above types and sample digital objects used in this tutorial here. You can then load this information using the Cordra UI. To do that, sign into Cordra as admin, select the Admin->Types dropdown menu, and click the "Load from file" button. In the dialog that pops up, select the types file you downloaded and check the box to delete existing objects. Click "Load" to import the types into Cordra. You can then view the digital objects in Cordra as they are discussed below.

If you view the type digital objects, you will see that the types have the hashObject property set to true. This means that the Cordra software will automatically generate hashes for the objects when they are created.

Let us say the id and hash for each of the three objects are as follows:

```
{
  "id": "test/macbeth",
  "hash": "ef8491742fe830636b952e457f168b38f61440b0d9ff8b473765e114721d63d"
}
{
  "id": "test/hamlet",
  "hash": "f4a9a2eb4b0b81dafa227fd80c4e824ad5966d789e23daa7054bd03eed8e37b22"
}
{
  "id": "test/othello",
  "hash": "60e7d4998d3bd9ae571c805a03a89e6273bb18883152a1f99a09b65750eface"
}
```
First, we create an initial block. Since this is the first block in the chain, the previousBlock pointer is omitted. Two objects are added to the pointers array, one for test/macbeth and one for test/hamlet.

Observe in the JSON of the initial Block object below that it contains a hashes section in the metadata:

```json
{
    "id": "test/first-block",
    "type": "Block",
    "content": {
        "pointers": [
            {
                "id": "test/hamlet",
                "hash": "f4a9a2eb4b0b81dafa227fd80c4e824ad5966d789e23daf7054bd03eed8e37b22"
            },
            {
                "id": "test/macbeth",
                "hash": "ef8491742fe830636b952e457f168b38f61440bddd9ff8b473765e1114721d63d"
            }
        ]
    },
    "metadata": {
        "hashes": {
            "alg": "SHA-256",
            "content": "705c6e8df77c08749a02466f40b75ca3ca4768c96ca6362fd862a116a62d3d66",
            "full": "c6ff828bc74b5d5ac25b7640dbf3c28d83f6db3643fe6441c5732f0765480518"
        }
    }
}
```

To create a second Block, we set its previousBlock pointer to point at the initial block, including the full hash of the pointed at object in this object. We also add in a pointer to test/othello:

```json
{
    "id": "test/second-block",
    "type": "Block",
    "content": {
        "previousBlock": {
            "id": "test/first-block",
            "hash": "c6ff828bc74b5d5ac25b7640dbf3c28d83f6db3643fe6441c5732f0765480518"
        },
        "pointers": [
            {
                "id": "test/othello",
                "hash": "60e7d4998d3bd9ae571c805a03a89e6273bb18883152a1f99a409b65750efacd"
            }
        ]
    }
}
```

(continues on next page)
The following type method on the Block type can be used to verify the chain to ensure that the pointed at objects have not been updated since the chain has been created.

```javascript
var cordra = require('cordra');
var cordraUtil = require('cordraUtil');

exports.methods = {};
exports.methods.verifyChain = verifyChain;

function verifyChain(block, context) {
    var report = cordraUtil.verifyHashes(block);
    if (!report.full) {
        throw "The hashes on this object " + block.id + " do not verify";
    }
    if (block.content.pointers) {
        for (var i = 0; i < block.content.pointers.length; i++) {
            var pointer = block.content.pointers[i];
            var pointedAt = cordra.get(pointer.id, true);
            if (pointedAt === null) {
                throw "The object " + pointer.id + " pointed at by " + block.id + " is missing";
            }
            var pointedAtReport = cordraUtil.verifyHashes(pointedAt);
            if (!pointedAtReport.full) {
                throw "The hash on the pointed at object " + pointer.id + " is invalid";
            }
            if (pointer.hash !== pointedAt.metadata.hashes.full) {
                throw "The full hash of pointed at object " + pointer.id + " does not match the hash stored in this block";
            }
        }
    }
    if (block.content.previousBlock) {
        var previousBlock = cordra.get(block.content.previousBlock.id, true);
        if (previousBlock === null) {
            throw "The previous block is missing";
        }
    }
}
```

(continues on next page)
throw "Previous block " + block.content.previousBlock.id + " referenced by " + block.id + " is missing";
  }
  var previousBlockReport = cordraUtil.verifyHashes(previousBlock);
  if (!previousBlockReport.full) {
    throw "The hash on the previous block " + previousBlock.id + " is invalid";
  }
  if (block.content.previousBlock.hash !== previousBlock.metadata.hashes.full)
  {
    throw "The full hash of previous block " + previousBlock.id + " does not match the hash stored in this block";
  }
  return verifyChain(previousBlock, context);
}

Invoking this type method will recursively verify the hashes for each pointer against the target object, following the pointers back to the previous block. Responding with true if all the hashes verify. If at any point a particular hash cannot be verified, the method will throw an error with a message indicating which pointer has failed.

Example Invocation:
```
curl -k -X POST 'https://localhost:8443/call/?objectId=test/second-block&method=verifyChain' \
-H "Content-Type: application/json" -H "Authorization: Bearer ACCESS_TOKEN"
```

### 6.7 OAI-PMH in Cordra

We created a proof-of-concept to demonstrate how digital objects in Cordra can be disseminated via OAI-PMH. This is not a generic application that automatically associates an OAI-PMH interface with any Cordra instance, but a specific demonstration that we had put together that could act as a template or a starting point for Cordra projects with OAI-PMH needs.

For this proof-of-concept, we make use of sample Paper and Book types, and enable OAI-PMH access to objects of those types. Paper type is a stand-in for an academic research paper, and Book type is a stand-in for published books. Here are a few examples of OAI-PMH calls that will work in your local Cordra instance once the proof-of-concept is enabled:
6.7.1 Example OAI-PMH URLs

6.7.1.1 Identify

http://localhost:8080/cordra-oai-pmh/?verb=Identify

6.7.1.2 ListIdentifiers

http://localhost:8080/cordra-oai-pmh/?verb=ListIdentifiers&from=2000-01-01T00:00:00Z&metadataPrefix=oai_dc

6.7.1.3 GetRecord

http://localhost:8080/cordra-oai-pmh/?verb=GetRecord&identifier=test/eb91eb599520927f9c8e&metadataPrefix=oai_dc

Here test/eb91eb599520927f9c8e is the id of a sample Cordra object. Replace it with an actual identifier if you are not using the sample data.

6.7.1.4 ListRecords

http://localhost:8080/cordra-oai-pmh/?verb=ListRecords&from=2000-01-01T00:00:00Z&metadataPrefix=oai_dc

http://localhost:8080/cordra-oai-pmh/?verb=ListRecords&resumptionToken=token_returned_from_a_previous_call

6.7.1.5 ListMetadataFormats

http://localhost:8080/cordra-oai-pmh/?verb=ListMetadataFormats

6.7.2 Design

Enabling OAI-PMH support within Cordra using our application requires two sets of capabilities:

- Metadata Translation: A capability that instructs the OAI-PMH interface how to convert Cordra objects into OAI-PMH expected metadata. We enabled this capability as a set of Type Methods that are configured on each type of Cordra object meant for OAI-PMH based dissemination. These methods, at runtime, read the information from a given Cordra object and produce bytes that are meaningful from a OAI-PMH dissemination standpoint. Details about these specific methods are discussed in the following sections.
• Protocol Interface: A capability that accepts OAI-PMH requests from users and invokes the aforementioned methods to produce the desired responses. We enabled this as a Java servlet application that should be appropriately configured to reach the Cordra it is fronting.

6.7.3 Setup

The sample objects and servlet application required are included in the Cordra download, in the extensions/oai-pmh directory.

6.7.3.1 Servlet Install

The Java servlet application war file should be placed in the data/webapps/ directory of the Cordra deployment. The data directory of the Cordra deployment should also contain a configuration file called oai-pmh-config.json.

Example configuration file:

```json
{
  "identity" : {
    "adminEmail" : "admin@example.com",
    "baseURL" : "http://localhost:8080/cordra-oai-pmh/",
    "repositoryName" : "Test"
  },
  "maxPageSize" : 2,
  "cordraBaseUri" : "http://localhost:8080/",
  "cordraTypes" : ["Book", "Paper"]
}
```

The identity information in the configuration file is used by the servlet application to respond to Identify verb requests.

The maxPageSize property is used by the servlet application when list requests, i.e., ListRecords and ListIdentifiers, are made. In the above example, it has been set to a very small number to demonstrate request resumption tokens in a system that only contains a handful of objects. In a production deployment this number should be set to something much larger.

cordraBaseUri is the URI to which the Java servlet application connects to interact with the Cordra HTTP API. Since the Java servlet application is run on the same environment as Cordra in our proof of concept, the above example refers to the Cordra instance deployed at the localhost.

cordraTypes lists the types of objects in the Cordra instance that should be made available to the OAI-PMH interface. For example, you might have objects of type Group or User that you do not want to expose over this interface. In the example deployment, two different types are used, Paper which represents the metadata of an academic research paper, and Book which represents the metadata for a published book.
6.7.3.2 Type Methods

The zip file download includes the necessary type object for our example application, in addition to a few sample objects. There is also a type called OaiUtil that contains some Javascript methods that are used by the Book and Paper types.

You can load the objects using the Cordra UI. Sign in into Cordra as admin and select the Admin->Types dropdown menu. Click the “Load from file” button. In the dialog that pops up, select the oai-pmh-objects.json file you downloaded and check the box to delete existing objects. Click “Load” to import the objects into Cordra.

To see examples of the methods described below, sign in as admin to the Cordra UI look at the Book and Paper types.

For details about how to write methods like these in Cordra, please refer to Type Methods.

6.7.3.3 Authorization

The type methods should be configured such that they can be publicly accessed (so that the servlet application can access them without any specific credentials). To do that, in the Admin->Authorization section of the Cordra UI, replace the default Authorization config with the following:

```
{
    "schemaAcls": {
        "Book": {
            "defaultAclRead": [
                "public"
            ],
            "defaultAclWrite": [
                "public"
            ],
            "aclCreate": [],
            "aclMethods": {
                "default": {
                    "instance": [
                        "public"
                    ],
                    "static": [
                        "public"
                    ]
                }
            }
        },
        "Paper": {
            "defaultAclRead": [
                "public"
            ],
            "defaultAclWrite": [
                "public"
            ]
        }
    }
}
```

(continues on next page)
This configuration enables public access to all instance and static methods on Book and Paper types.
6.7.3.4 Metadata Translation using JavaScript

The proof of concept enables five OAI-PMH request verbs: Identify, ListIdentifiers, GetRecord, ListRecords, and ListMetadataFormats.

- **Identify**: The information necessary to respond to the Identify verb is already configured with the servlet application (see next section for details). As such, no specific support for this verb at Cordra is needed.

- **ListIdentifiers**: Cordra's REST API already supports listing identifiers of Cordra objects of a given type. As such, no specific Type method is required to support this verb.

- **GetRecord**: OAI-PMH expects a metadata description for each object. At the minimum for such description, OAI-PMH expects a DC format, which is short for XML serialization of metadata that uses properties from Dublin Core specification. OAI-PMH allows metadata descriptions in other formats. Because information in Cordra objects need not be readily available for dissemination per OAI-PMH, Type methods are required to translate Cordra objects to meet OAI-PMH needs. One method getAsXml should be associated with each type of Cordra object. That method should have instructions to convert information from the Cordra object into the desired XML. If the intention is to just produce XML that conforms to DC format, a convenience method getAsDublinCoreJson can be used instead. This method expects a JSON object containing various DC fields. The OAI-PMH servlet application is hard-wired to execute either of these methods depending on the user request. Example code for these two methods is listed in the Appendix.

- **ListRecords**: ListRecords is, in effect, a listing of all records where each record corresponds to the GetRecord response for a given object. As such, the methods discussed above are sufficient for the servlet application to fulfill user requests that correspond to this verb.

- **ListMetadataFormats**: As mentioned above, metadata formats other than DC can be supported by an OAI-PMH interface. Because Cordra objects are typed, each type should indicate the supported metadata formats. A method listMetadataFormats should be used to return a list of supported formats. Example code is listed in the Appendix.

6.7.3.5 Protocol Interface Servlet Application

A Java servlet application was developed for this proof of concept that performs three actions:

- It accepts valid OAI-PMH requests from users,

- It invokes appropriate Cordra APIs (and in turn invokes appropriate Type methods wherever applicable) to retrieve information for a given user request, and

- It serializes the retrieved information to the desired form and responds back to the users.
Cordra Technical Manual, Release 2.5.2
6.7.3.6 Resumption Token
Resumption token, according to OAI-PMH, is a token that enables clients to retrieve the next set
of results for a given request when the total number of results are deemed large enough to require
multiple responses. The resumption token is implemented as a stateless string: a base64 encoding
of the bytes of a JSON string that when decoded informs Cordra the next set of Cordra objects that
should be returned. For example, the decoded information may look like this:
{"txnId":1557422842749010,"from":"2000-01-01T00:00:00Z","metadataPrefix":"oai_dc"}

6.7.4 Appendix
Metadata for the OAI-PMH results are generated at runtime using Type methods. In particular, the
Java servlet application knows to invoke speciﬁc methods that are attached to the applicable types.
Those methods are getAsXml, getAsDublinCoreJson, and listMetadataFormats.
6.7.4.1 getAsXml
This method expects two arguments: the instance of the Cordra object and a context object. In
this case, the context object will contain a string property context.params.format. The value of this
property will contain the metadataPreﬁx of the originating OAI-PMH call. Your code should branch
on the value of this property and construct an XML string in the requested format. The function
should return that XML string. If you do not want this type of object to support the requested
format you should throw an exception like this:
var format = context.params.format;
if ("oai_custom_1" !== format) throw "Format not supported";
If your object does support the requested format, construct an XML string and return it. An example
is shown below:
var paper = cordraObject.content;
var xml = '<paper>';
xml += '<identifier>'+ cordraObject.id +'</identifier>';
if (paper.title) {
xml += '<title>'+paper.title+'</title>';
}
if (paper.abstract) {
xml += '<abstract>'+paper.abstract+'</abstract>';
}
xml += '</paper>';
return xml;
In the example on the Book type, an included utility function oaiUtil.toXml() is used to generate
the XML from the JSON of the object automatically.

OAI-PMH in Cordra

287


var book = cordraObject.content;
var xml = oaiUtil.toXml("book", book);

Note that oaiUtil.toXml() is implemented for this proof-of-concept; as implemented, this method toXml() does not support attributes, and implements one specific way of representing lists of elements. It is included here solely to provide a starting point, one way in which a developer might choose to generate their specific XML records.

### 6.7.4.2 getAsDublinCoreJson

The XML serialization of Dublin Core metadata is a known structure to the servlet application. As such a utility is provided that can automatically convert a specific JSON object with correctly named properties into a Dublin Core XML record. The return value of the method getAsDublinCoreJson should be a JSON object with property names that match the 15 terms from the Dublin Core set:

```javascript
```

You can either include a single top level property, or if you want to include more than one property of that type, you can use an array with the same name like in this example:

```javascript
var dcJson = {
  "title" : "foo",
  "creator" : ["Bob", "Alice"]
};
```

The Java servlet application (cordra-oai-pmh.war) will automatically convert the returned JSON object into Dublin Core compatible XML including the dc: prefix to the property names.

An additional utility method oaiUtil.mapTo() can be used to map the values on the input Cordra object to the desired output properties. It takes a map from output property name to path of input property, like this:

```javascript
var map = {
  "subject" : "/genre",
  "creator" : "/authors",
  "publisher" : "/publishedBy",
  "language" : "/language",
  "date" : "/publicationDate"
};
var dcJson = oaiUtil.mapTo(cordraObject, map);
```
6.7.4.3 listMetadataFormats

For each type of digital object that you are using with OAI-PMH, the metadata formats must be explicitly listed. This is done by implementing and exporting the following JavaScript method, like this:

```javascript
exports.staticMethods = {};  
exports.staticMethods.listMetadataFormats = listMetadataFormats;

function listMetadataFormats() {
    return [
        {
            "metadataPrefix" : "oai_dc",
            "schema" : "http://www.openarchives.org/OAI/2.0/oai_dc.xsd",
            "metadataNamespace" : "http://www.openarchives.org/OAI/2.0/oai_dc/"
        },
        {
            "metadataPrefix" : "oai_custom_1"
        }
    ];
}
```

6.8 Sending Emails

Cordra can be configured to allow sending emails from **Type Methods**. In order to enable this, you will need to add the supporting jar libraries to your Cordra instance and then configure the emails you wish to send.

6.8.1 Installation

For this example, we’ll use the Simple Java Mail emailing library to send emails. The jar files needed are included in the Cordra distribution, in the extensions/user-registration/lib directory. To install this library, put these jar files into a directory called `lib` in your Cordra data directory. If the `lib` directory doesn’t exist, create the `lib` directory in the Cordra data directory.

You will need to restart the Cordra server once the Java Mail Client library is installed if your Cordra server is already running.
6.8.2 SMTP Configuration

Create a file in the data directory called emailServerConfig.json with the following contents:

```json
{
    "serverAddress": "smtp.example.com",
    "serverPort": 587,
    "enableStartTls": true,
    "trustHosts": "smtp.example.com",
    "username": "your-smtp-username",
    "password": "your-smtp-password"
}
```

Be sure to replace the example settings with the settings for your actual SMTP server. Finding these settings is outside of the scope of this document, but most email providers have some way of sending emails using SMTP. You can also use Amazon’s Simple Email Service for testing.

6.8.3 Email Templates

In the data directory, create an emailTemplates directory. In that directory, create files called creation.txt and creation.html. These files should contain the text and html (respective) that should be sent in the email.

6.8.4 Sending Emails

As an example, we’ll modify the beforeSchemaValidation method on the User object to send an email whenever a new User object is created. Replace the default User schema javascript with the following:

```javascript
exports.beforeSchemaValidation = beforeSchemaValidation;

var emailConfig = {
    "toAddress": "test@example.com",
    "fromAddress": "admin@example.com",
    "subject": "testing javascript email"
};

function beforeSchemaValidation(obj, context) {
    if (context.isNew) {
        sendEmail(emailConfig.toAddress);
    }
    return obj;
}

function sendEmail(toAddress) {
    // Java types
    (continues on next page)
Next, try creating a new User in Cordra. You should receive an email containing the message specified in the JavaScript configuration.

The email templates are loaded from the files we created before (lines 24-25). Be sure to modify emailConfig with real email addresses to use for sending the email. Email server config is loaded...
from a local file in the `loadServerConfig` function (lines 47-50), so the secrets are never network accessible through Cordra.

We are using `context.isNew` to make sure we only create a token on new object creation (line 10). The Java classes from Simple Mail Client are loaded into the JavaScript (lines 18-20) and then used using the Simple Java Mail API (lines 26-44).

### 6.9 External Authentication Provider

Cordra supports an `authenticate` JavaScript lifecycle hook that provides complete control how a request is authenticated. This section provides a complete example of one way in which this hook can be used to authenticate a JWT (Json Web Token) that have been supplied by a 3rd party authentication service. Below is an example of the decoded payload section of such a token:

```json
{
    "username": "fred",
    "groupIds": [
        "test/group1"
    ],
    "iss": "https://example.com/",
    "sub": "60afe42cebfb9c0068924e85",
    "aud": [
        "https://cordra.org/",
        "https://example.com/userinfo"
    ],
    "iat": 1630609934,
    "exp": 1630696334,
    "azp": "0s5qpVcOmDHuzowxcmI2fU7fRFxNyw0W",
    "scope": "openid profile email address phone",
    "gty": "password"
}
```

The below example `authenticate` hook implementation on the design object roughly follows these steps:

1. Load the providers public key in JWK format from the `cordra data` directory.
2. Check if authInfo contains an correctly formatted token from the external provider.
3. This is done by first checking if the token is a JWT.
4. Then decoding the JWT and checking that the `iss` is who we expect it to be.
5. If it is not an authentication attempt with a suitable token we return null.
6. If we have an suitable token we verify it against the public key of the provider.
7. Check the values of the claims stored in the token to make sure they are correct.
8. Assuming those tests pass, return an object containing the userId, username, groupIds and the expiration from the token.
const cordra = require('cordra');
const cordraUtil = require('cordraUtil');

let providerPublicKey = null;

exports.authenticate = authenticate;

function authenticate(authInfo, context) {
  cacheKeyIfNeeded();
  if (isTokenAuthentication(authInfo)) {
    return checkCredentials(authInfo);
  } else {
    return null;
  }
}

function isTokenAuthentication(authInfo) {
  if (authInfo.token) {
    if (isJwtFromProvider(authInfo.token)) {
      return true;
    }
  }
  return false;
}

function isJwtFromProvider(token) {
  if (!token.includes('.')) {
    return false;
  }
  try {
    const claims = cordraUtil.extractJwtPayload(token);
    return "https://example.com/" === claims.iss;
  } catch (error) {
    return false;
  }
}

function checkCredentials(authInfo) {
  const token = authInfo.token;
  const payload = cordraUtil.extractJwtPayload(token);
  const isVerified = cordraUtil.verifyWithKey(token, providerPublicKey);
  const claimsCheck = checkClaims(payload);
  const active = isVerified && claimsCheck;
  const result = {
    active: active
  };
}

(continues on next page)
if (active) {
    result.userId = payload.sub;
    if (payload.username) {
        result.username = payload.username;
    }
    if (payload.groupIds) {
        result.groupIds = payload.groupIds;
    }
    if (payload.exp) {
        result.exp = payload.exp;
    }
    result.grantAuthenticatedAccess = true;
}
return result;
}

function isBasicAuth(authHeader) {
    return authHeader.startsWith("Basic ");
}

function isBearerTokenAuth(authHeader) {
    return authHeader.startsWith("Bearer ");
}

function getTokenFromAuthHeader(authHeader) {
    return authHeader.substring(authHeader.indexOf(" ") + 1);
}

function checkClaims(claims) {
    if (!claims.iss || !claims.exp || !claims.aud) {
        return false;
    }
    if ("https://example.com/" !== claims.iss) {
        return false;
    }
    const nowInSeconds = Math.floor(Date.now() / 1000);
    if (nowInSeconds > claims.exp) {
        return false;
    }
    const aud = claims.aud;
    if (!checkAudience(aud)) {
        return false;
    }
    return true;
}
function checkAudience(audElement) {
    let aud = [];
    if (typeof audElement === "string") {
        aud.push(audElement);
    } else if (Array.isArray(audElement)) {
        aud = audElement;
    } else {
        return false;
    }
    if (aud.includes("https://cordra.org/")) {
        return true;
    } else {
        return false;
    }
}

function cacheKeyIfNeeded() {
    if (!providerPublicKey) {
        const configDir = getDataDir();
        const File = Java.type('java.io.File');
        const keyPath = configDir + File.separator + "publicKey.jwk";
        providerPublicKey = readFileToJsonAndParse(keyPath);
    }
}

function getDataDir() {
    const System = Java.type('java.lang.System');
    const cordraDataDir = System.getProperty('cordra.data');
    return cordraDataDir;
}

function readFileToString(pathToFile) {
    const path = Java.type('java.nio.file.Paths').get(pathToFile);
    const string = Java.type('java.nio.file.Files').readString(path);
    return string;
}

function readFileToJsonAndParse(pathToFile) {
    const jsonString = readFileToString(pathToFile);
    const result = JSON.parse(jsonString);
    return result;
}
6.9.1 Customizing the UI to use an external authentication provider

The previous section describes how to verify an access token provided by an external authentication provider on the server side. You may also customize the Cordra UI authentication dialog to sign in with a 3rd party. This is done by embedding in an iframe an html page that you need to write. The content of that html page is entirely under your control. A typical example of such a page would be to load a 3rd party JavaScript authentication library, render a sign in button and acquire the access token. This html page communicates with the parent Cordra UI using `postMessage`. In particular when your custom page wants to send the newly acquired access token to the Cordra UI it should send the following via `postMessage`:

```javascript
let message = {
    type: "customAuthentication",
    token: accessToken
};
window.parent.postMessage(message, '*');
```

Your custom page should also listen for a sign out message:

```javascript
window.addEventListener("message", (event) => {
    let message = event.data;
    if (message && message.type === "signOut");
    // handle sign out with 3rd party provider
});
```

A complete example of a Cordra UI customization html page using the Sign in with Google JavaScript API is shown below:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Test google sign in</title>
    <div id="buttonDiv"></div>
</head>
<body>
<script src="https://accounts.google.com/gsi/client" async defer></script>
<script>
    const CLIENT_ID = "your client id provided by google";

    window.onload = function () {
        window.addEventListener("message", (event) => {
            let message = event.data;
            if (message && message.type === "signOut");
            google.accounts.id.disableAutoSelect();
        });
    }
</script>
</body>
</html>
```

(continues on next page)
Once written your custom authentication page should be stored as a payload on a Cordra object. Typically the design Cordra object is used to store this payload with the payload name “customAuthentication”.

To configure the UI to use this custom authentication page add the follow JSON object to the Cordra UI config as a top level property:

```
"customAuthentication": {
   "url": "/objects/design?payload=customAuthentication",
   "tabName": "Google",
   "height": 100
}
```

The url points at the custom html page that is stored as a paylaod on the design object.
The iframe will be created in a new tab in the authentication dialog. You can customize the name of that tab with the tabName property.
The height property allows you to control the height of the iframe in the new tab.
6.10 Collab Prototype Software

Collab is an open source prototype software extension to Cordra software that can be integrated into existing data management systems to augment those systems with a capability to allot digital object identifiers to data and answer certain questions that demonstrate the readiness of the data for use in science and decision-making. One of the core functionalities of the software is to demonstrate how to easily create, repeat, and reproduce complex scientific workflows.

Collab is described in more detail at this [website](#). High availability and scalability are not specifically addressed in this prototype software.

The prototype software is made available in the Cordra download, in the `extensions/collab` directory.

This project is based upon work supported by the National Science Foundation under Award Number 1838981. The two resulting research papers are made available [here](#) and [here](#).

6.11 Neo4j as an Additional Index

The type of index used by a system dictates the sorts of queries that can be run. Cordra's default index can be configured with a selection of Lucene-based indexes, however there are circumstances that requires something else. In such a situation an additional index can be added to Cordra by inserting items into that index in response to Cordra's lifecycle hooks.

This example shows how to install an extension that allows Cordra to insert data into the graph-based index Neo4j. A graph-based index allows for certain types of queries over highly connected data that are not possible in a single pass with other indexing systems.

This project contains lifecycle hooks written in Java that will create nodes and relationships in a Neo4j graph database. The nodes created correspond to Cordra objects and the relationships are determined by properties on the Cordra objects that are defined as handleReference in the JSON Schema for that type of object.

6.11.1 Installation Instructions

You can download a desktop installation for Neo4j here: [https://neo4j.com/download/](https://neo4j.com/download/)

On the version of the Neo4j desktop app we tested it was necessary to reset its DBMS password before it would allow clients to connect. To do this run the neo4j desktop app. Then select "Movie DBMS" from the "Example Project". This will show information about the DBMS in the right panel. In that right panel click the "Details" tab. In the "Details" tab enter a new admin password in the "Reset DBMS password" section.

Ensure the sample database is empty using:

```
MATCH (n) DETACH DELETE n
```

Obtain the file `cordra-neo4j-test-1.0.0.jar` from the Cordra download distribution in the `extensions/neo4j-test` directory. Then add it as a payload with the payload name "java" to the
design object in Cordra. You can see the design object by signing in to Cordra as admin and then from the menu select Admin->Design Object... Edit the design object and scroll all the way down to add payloads.

Also add a payload containing a config.json file named “neo4jConfig” to the design object. An example config.json is below (and also in the extensions/neo4j-test directory):

```json
{
    "user": "neo4j",
    "password": "password",
    "uri": "bolt://localhost:7687",
    "databaseName": "neo4j",
    "propertyNameMode": "topLevel",
    "excludeTypes": ["User", "Group", "CordraDesign", "Schema"],
    "includeTypes": ["Person", "Movie"]
}
```

The properties “user”, “password” and “uri” are required.

The property “databaseName” is optional and will default to the value “neo4j” if missing.

The property “propertyNameMode” configures the method used to generate the node properties from the cordra object properties. It may take the string values “topLevel” or “jsonPointer”. The value “topLevel” is default if missing.

“topLevel” takes the top level properties of the Cordra object content that have primitive values and sets those as the Neo4j node property names.

“jsonPointer” will create a JSON Pointer for each of the properties of the cordra objects content and set those JSON Pointers as the Neo4j node property names.

Note that in both cases the created node will be given the properties “_type” and “_id”.

The properties “excludeTypes” and “includeTypes” can be used independently or together to control objects of which Cordra types are created as nodes in Neo4j.

“excludeTypes” can be used to specify types that should not be created in Neo4j. If missing no types will be explicitly excluded.

“includeTypes” can be used to specify types that should be created in Neo4j. If missing all types not listed in “excludeTypes” will be created.

“verbose” optional boolean with default value false. When set to true queries sent to Neo4j will be logged to standard out.

To demonstrate the functionality of this project create the types “Movie” and “Person” in Cordra. The JSON Schemas for those types are in the extensions/neo4j-test directory. You could then create a Movie instance “Top Gun” and then create a Person instance “Tom Cruise”, on that person object add the “Top Gun” movie to the “ACTED_IN” array.

The example comes with a class MoviesImporter which will import many actors and movies into Cordra. Run the script import-movies in the extensions/neo4j-test directory with the base URI, username, and password for your running Cordra instance, for example:
Some sample Neo4j queries to try are in the file sample-queries.txt in the extensions/neo4j-test directory.

### 6.11.2 Neo4j Browser Styling

In order to get the Neo4j Browser to display Movie and Person objects with useful captions, you can create a file “style.grass” and drag it onto the Neo4j Browser window; an example is below and also in the extensions/neo4j-test directory:

```plaintext
node {
  diameter: 50px;
  color: #A5ABB6;
  border-color: #9AA1AC;
  border-width: 2px;
  text-color-internal: #FFFFFF;
  font-size: 10px;
}
relationship {
  color: #A5ABB6;
  shaft-width: 1px;
  font-size: 8px;
  padding: 3px;
  text-color-external: #000000;
  text-color-internal: #FFFFFF;
  caption: "<type>";
}
node.CordraObject {
  color: #F79767;
  border-color: #F36924;
  text-color-internal: #FFFFFF;
  defaultCaption: "<id>";
  caption: "{title}";
}
node.Person {
  color: #57C7E3;
  border-color: #23b3d7;
  text-color-internal: #2A2C34;
  defaultCaption: "<id>";
  caption: "{name}";
}
node.Movie {
  color: #F16667;
  border-color: #eb2728;
  text-color-internal: #FFFFFF;
}
```
6.11.3 Node Labels

All Neo4j nodes will be given the labels “CordraObject” and a label that corresponds to its type.

6.11.4 Relationships

This project will automatically create the relationships between nodes in Neo4j. It does this by inspecting the JSON Schema of the Cordra object that is being created or updated. Any properties on the content of the Cordra object that have a corresponding JsonSchema property `cordra.type.handleReference` will be treated as a relationship in neo4j.

The name of the relationship will come from the name of the property on the Cordra object. If the property is deep in a nested JSON structure the JSON Pointer is processed from right to left taking the first non-numeric part as the relationship name. e.g. if you had a JSON object:

```
{
    "ACTED_IN": [ 
        "test/123"
    ]
}
```

The string value “test/123” is a pointer to another Cordra object. The jsonPointer to this value is “/ACTED_IN/0” so the resulting relationship name used in Neo4j will be “ACTED_IN”.

The Cordra Neo4j indexer gives all nodes in Neo4j the label “CordraObject”. As a result of this you can see the graph of all CordraObjects with the Neo4j query:

```
MATCH (n:CordraObject) RETURN n;
```

6.11.5 Custom Operations

When you add the jar as a “java” payload to the design object a number of static operations (aka static type methods) will be added to the design object. You can see these in the Cordra UI by going to the design object by selecting Admin->Design Object… then click the “Methods” tab:

“reloadNeo4jConfig”: If you invoke this method the “neo4jConfig” JSON payload will be read and the connection to Neo4j will be reinitialized using that configuration. This allows you to change the configuration payload without needing to restart Cordra.

“deleteInNeo4j”: Accepts an “id” attribute for the single object that should be deleted from the Neo4j index. Note that the delete operation will fail if there are other nodes that have relationships pointing at the node being deleted.
"deleteAllInNeo4j": Removes all Cordra object nodes and relationships from the Neo4j index.

"searchNeo4j": Accepts a "query" attribute that is expected to contain a query in the Cypher query syntax. This query is passed to the Neo4j index and the full response is returned as JSON.

### 6.11.6 Reindexing Objects in Neo4j

Neo4j does not permit creating relationships if the nodes on both sides of the relationship do not exist. For example if you have A->B and you try and index A before indexing B the operation will fail. The same is true if you have A->B and B->A. The solution to this is to index objects in two passes, first to create the nodes in the index and then a second pass to create the relationship.

This project provides a number of custom operations that allow you to instruct Cordra to reindex its objects in Neo4j. These operations will either perform the reindex in two passes, indicated by the name of the operation, or they will accept a boolean attribute "includeRelationships" which defaults to true. If you are using one of the operations that accepts "includeRelationships" you will need to invoke the operation twice, first with "includeRelationships":false and then again with "includeRelationships":true.

"reindexAllInNeo4jTwoPass": Reindexes all Cordra objects into Neo4j in two passes, first just the nodes and then the nodes with relationships.

"reindexAllInNeo4j": Reindexes all Cordra objects into Neo4j in a single pass. Accepts a boolean attribute "includeRelationships" which defaults to true.

"reindexQueryResultsInNeo4jTwoPass": Accepts a "query" attribute. This is a Lucene query that is applied to Cordra index. The results are indexed into Neo4j in two passes, first just the nodes and then the nodes with relationships.

"reindexQueryResultsInNeo4j": Accepts a "query" attribute. This is a Lucene query that is applied to Cordra index. The results are indexed into Neo4j in a single pass. Accepts a boolean attribute "includeRelationships" which defaults to true.

"reindexOneInNeo4j": Accepts an "id" attribute and boolean attribute "includeRelationships". This operation will reindex the object specified in the "id" attribute.

### 6.12 Partial Replication/Aggregation

This project demonstrates how to install a module into Cordra that performs partial replication/aggregation from other Cordra servers.

This is pull-based replication in that the destination Cordra server is the one doing the work of pulling objects from one or more remote Cordra servers. Which objects are pulled is highly configurable, types of objects can be explicitly included or excluded. In addition a custom query can be supplied to filter which objects are pulled. One advantage of this pull-based approach is that it allows the destination Cordra server to have complete control over which objects are replicated, and when they are replicated. This can be useful in cases where the destination Cordra server has limited resources, or if the network connection between the two Cordra servers is unreliable.
Replication in this project is performed by using the existing Cordra search API. In short the pulling Cordra periodically searches each remote Cordra for new or updated objects that match the constructed query. This is made possible because each object contains a transaction id and that transaction id is indexed and searchable. As such the pulling Cordra can record the highest transaction id it has seen so far and on the next poll include a range search from that highest seen transaction id to the latest transaction id.

### 6.12.1 Installation Instructions

Note that installation and configuration is only performed on the destination Cordra instance. This instance will pull objects from the source instance. From the source Cordra’s perspective it is responding to ordinary search and retrieve requests. It may be necessary to add a user account and access control on the source instance.

1) Obtain the file `cordra-replication-1.0.0.jar` from the Cordra download distribution in the `extensions/replication` directory.

2) Create a new type in Cordra called `CordraReplicationState`.

The JsonSchema for this type is:

```json
{
    "type": "object",
    "properties": {
        "serviceId": {
            "type": "string"
        },
        "pollingIntervalSeconds": {
            "type": "number"
        },
        "services": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "id": {
                        "type": "string"
                    },
                    "baseUri": {
                        "type": "string"
                    },
                    "ipAddress": {
                        "type": "string"
                    },
                    "port": {
                        "type": "number"
                    },
                    "username": {
                        "type": "string"
                    }
                }
            }
        }
    }
}
```

(continues on next page)
3) Attach the jar file as a payload to the CordraReplicationState Schema object. You must name that payload “java”.

4) Configure access control for the type CordraReplicationState such that only your admin user
can read and write to objects of that type. This is important as instances of CordraReplica-
tionState contain the credentials used to connect to other Cordra instances.

5) Create an instance of the CordraReplicationState type. Give it the following values - change
the details as needed:

```
{
    "pollingIntervalSeconds": 60,
    "services": [
        {
            "id": "test/service",
            "baseUri": "https://10.0.1.157:8443/doip/",
            "ipAddress": "10.0.1.157",
            "port": 9000,
            "username": "admin",
            "password": "password",
            "protocol": "DOIP",
            "excludeTypes": [
                "Schema"
            ]
        }
    ]
}
```

"pollingIntervalSeconds": Number of seconds to wait between the end of one replication process
and starting the next.

The "services" property holds an array of objects where each object describes how to connect to
another Cordra service that this Cordra will be pulling from.

"id": The id of the service being pulled from.

"baseUri": The base URI of the remote Cordra to use if connecting via DOIP-HTTP or CORDRA-
HTTP protocols.

"ipAddress": The ip address of the remote Cordra to use if connecting via the binary DOIP protocol.

"port": The port of the remote Cordra to use if connecting via the binary DOIP protocol.

"username": The username to use when talking to this remote Cordra.

"password": The password to use when talking to this remote Cordra.

"protocol": The protocol to use when talking to this remote Cordra. May be "DOIP", "DOIP-HTTP",
"CORDRA-HTTP".

"excludeTypes": Types of Cordra objects to be be excluded when pulling from this remote Cordra.

"includeType": (Optional) Types of Cordra objects to be be excluded when pulling from this remote
Cordra.

"customQuery": (Optional) A custom query that will be ANDeed into the replication query when
pulling new objects.
6.12.2 Behavior

In order to demonstrate the replication behavior:

1. Run a source Cordra.

2. Run a destination Cordra, configured with CordraReplicationState as above, with the source Cordra a service to be pulled from.

Any objects (except of types excluded by the configuration in the destination Cordra’s CordraReplicationState) which are created or updated in the source Cordra will be replicated into the destination Cordra, after at most the time interval in the configured pollingIntervalSeconds.

6.12.3 Custom Operations

The CordraReplicationState type comes with two custom operations, “start” and “stop”. These can be used to pause and resume replication while Cordra is running.
7.1 Import and Export Tools

Import and export tools are included in the Cordra distribution. The export tool enables you to extract the digital objects from a given Cordra setup (single instance or a distributed system) as files into the environment from which the tool is run. The import tool enables you to ingest the output of an export tool process as digital objects into any Cordra setup. The rest of this section describes the specifics of these tools.

These tools do not use the Cordra API, but rather use the Cordra storage module to interact with the underlying storage system. As a result, they are able to copy objects in and out of Cordra in their entirety, without modifying their contents in any way.

The import and export tools can be found in the /WEB-INF/tools directory after unzipping cor dra.war. To run the scripts, make sure the scripts are executable; if necessary, change file access permissions (e.g., by using a command like “chmod +x” on *nix systems).

Warning: Since these tools talk directly to storage they bypass all of the usual validation checks that Cordra makes. For example, during import, objects that do not match schemas can be inserted. Likewise, type objects or user objects with duplicate names can be inserted. Doing so could result in unexpected and/or unwanted behaviour. Cordra should be shutdown before you attempt an import or an export in order to curtail any parallel administrative activity.

When exported, files are produced that represent Cordra objects. Each file contains information from a corresponding Cordra object, and includes a JSON map of any payloads; and those payloads are encoded as base64 strings. Metadata and schema-driven information of each Cordra object is represented as JSON in those files. If wholesale changes to digital objects are required, it is easy to edit the representative files while in the export format, and subsequently import them into Cordra. However, because payloads are encoded as base64 strings, editing payloads while in the export format is not straightforward.

Once you have imported the objects, Cordra should be re-indexed to function properly. How you reindex depends on the type of indexer you are using. Deleting the existing index and restarting Cordra will trigger a reindex. For example, if you are using the default index that comes with Cordra, you can simply remove the data/cordraIndex directory. For related details, see Reindexing.

The specific commands to import and export are described next.
7.1.1 Export

An example to export digital objects from a local file system based Cordra:

```
./export-tool -c path/to/Cordra/config.json -d path/to/Cordra/data/folder/ -o path/to/folder/of/Cordra/Objects/ --tree --number-of-threads 24 --progress
```

The `--c` option is required, as config.json includes the details of the storage being used. If the storage specified in config.json uses the filesystem, the `--d` option is required. (It can be omitted for MongoDB and Amazon S3 storage, for example)

Either the `--o` option (to export to files) or `--s` option (to export to newline-delimited JSON on stdout) is required. If the `--o` option is provided, an additional `--tree` option may be provided to arrange exported Cordra Objects in a directory tree based on the hashes of the IDs of the Cordra Objects.

A `--number-of-threads` option may be used to indicate the number of threads exporting Cordra objects. The default number of threads used to export Cordra Objects is 1; export to a file system is generally I/O bound.

With the `--progress` option the tool reports a count of Cordra Objects exported after each export of a Cordra Object completes.

To run the import switch using a command-line interface, execute a command like the following.

An example to export from a backend system such as MongoDB or Amazon S3 (with their coordinates in config.json):

```
./export-tool -c config.json -o objects
```

7.1.2 Import

An example to import digital objects into a local file system based Cordra:

```
./import-tool -c path/to/Cordra/config.json -d path/to/Cordra/data/folder -i path/to/folder/of/Cordra/Objects/ --number-of-threads 32 --delete-design-first --delete-all-first
```

Either the `--i` option (to import from files in a directory) or the `--s` option (to import from newline-delimited JSON on stdin) is required. If `--s` is used, input must consist of JSON objects, one on each line, where each JSON object is free of newline characters and carriage return characters.

If the `--delete-design-first` option is used, the design object will be deleted before import.

If the `--delete-all-first` switch is used, all objects except the design object will be deleted before import.

An example to import digital objects into a backend system such as MongoDB or Amazon S3 (with their coordinates in config.json):

```
./import-tool -c config.json -i objects
```
7.1.3 Hashed directory output

With large numbers of objects, you may exceed the maximum number of files that your file system allows in a single directory. In such a case you can use the -t or --tree option.

```
./export-tool -c cordra/data/config.json -d cordra/data/ -o objects -t
```

This will hash the object ids and break that resulting hash into segments that are used to create a directory tree. The import tool does not need to have specified whether the import is from a hashed directory or not; it will work either way.

7.1.4 Limit the objects that are exported by id

```
./export-tool -c cordra/data/config.json -d cordra/data/ -o objects -i 123/abc -i 123/xyz
```

Here multiple -i arguments can be passed to the tool to specify which objects to export.

If you have a large number of objects you want to explicitly export, you can list their ids in a new line separated file and use the -l option.

```
./export-tool -c cordra/data/config.json -d cordra/data/ -o objects -l ids.txt
```

An additional tool called ids-by-query can be used to generate an ids file by running a query. Unlike export-tool and import-tool, it needs to access a running Cordra. This tool comes with the Cordra distribution and can be found in the bin directory.

```
./bin/ids-by-query -b http://localhost:8080 -u <username> -p <password> -o ids.txt -q <query>
```

7.1.5 Piping export to import

Instead of writing objects to files in a directory the export tool can instead write objects as newline-delimited JSON to standard out using the -s option. This can be piped to the import tool in a *nix environment.

```
./export-tool -c cordra/data/config.json -s | ./import-tool -c cordra2/data/config.json -s
```