AONS User Guide

Application Site Map

- Main Page / Tasks: Displays currently active tasks in the system
- Repositories: Overview of all repository connections in the system
    - Create a new Repository Connection
    - Repository Details: View details of a repository connection
        - Update connection details
        - Delete the Repository Connection
        - Update/Create/Delete crawl schedule
        - Cancel currently active crawl
    - Go to the repository explorer for this
- Registries: Overview of all registry connections in the system
    - Create a new registry connection
    - Registry Details: View details of a registry connection
        - Update registry connection details
        - Delete registry connection
        - Update/Create/Delete synchronize schedule
        - Cancel currently active synchronize

- Repository Explorer: View the summary of formats found during the crawl
    - Drill down through collections
    - Find unidentified formats and identify them
- Global Format Summary: View the formats found within this AONS instances repositories and their associated risk
    - Identify unknown formats
    - Perform risk assessments on the found formats
- Configuration
    - Configure Proxy Server Details
    - Configure Mail Server Details
- Notifications
    - Create new email destinations
- Tools: View all currently known “useful tool” link within the application
    - Create new Useful Tool
    - Delete a Useful Tool
    - Update a Useful Tool

Getting Started

Okay, so you've been told about AONS, what next?

The first step towards using AONS is downloading it. Currently, it can be found under the AONS Sourceforge project under file releases. After downloading the next step is to install AONS, this is covered in the Installation Guide, found under the docs folder of the project.

So, you've installed AONS, and it's not working for some reason. If AONS isn't loading, chances are probably on a deployment problem which can be found in the server log files. Have a look for any stack traces, and see if there are any answers to be found either on bugs already found with AONS on the Sourceforge project or on the Internet abroad. Most of the time, if you're deploying onto a known configuration (Tomcat + PostgreSQL) bugs should be limited. However, if you are deploying to a different configuration you may find a few issues which we will try and address; deploying to Oracle was perhaps the most painful experience so far due to it's table name restrictions (31 characters!!), though even the semi-large refactoring needed here was just a case of fixing in the daily build iterations.

AONS is currently beta software

AONS is still Beta software… we're moving towards a completely supported release, but for the moment, we will continue to make whatever changes are deemed necessary and future installs may not have a guaranteed upgrade path from a previous version. This was true when we changed the table names for Oracle, rather then provide migration scripts, we just concentrated on getting the change implemented and past us.

Once the user base grows and people are happy with the functionality in general, we will "downshift" into a bit more of a slower mode and ensure that future revisions are largely backwards compatible with clear migration paths if not.

Configuration Task Breakdown

Now the next bit: you've installed AONS and it is working. What next? I think we can describe the required AONS functionality in three broad areas:

1. System Administration Tasks (setting up connections to repositories)
2. Occasional tasks
3. Core tasks which will be the mainstay of daily tasks.

Runtime vs Static Configuration

AONS favours runtime configuration over backend XML configuration where possible. You will find that you should have almost no reason to ever go into the underlying XML configuration files for the system – whenever you are, you are either changing an element of configuration which we either deemed 90% of people wouldn't need to change or we've failed to make something into a runtime "knob" to turn.

Having configuration accessible at runtime has many advantages over static configuration – the main one being able to change the applications behaviour dynamically.

Proxy Server Configuration

In order to connect to some of the things mentioned here, there is a good chance you'll need to first configure your proxy server for the deployed instance of AONS.

Number	Step	Screenshot
1	Go to configuration side bar	
2	Click on the "Edit" configuration for the HTTP Proxy Settings	
3	Choose "Yes" to elect to use a proxy server for external HTTP connections. Click Next.	
4	Enter in the proxy server and port address, click next.	
5	Enter whether or not this proxy requires authentication. Click Next. If your proxy does not require authentication, skip to step 7.	
6	If your proxy does require authentication, enter the following details: - Proxy "Realm" - Proxy Username - Proxy Password Click "Next"	
7	Review and accept changes or go back and change your answers.	

Mail Server Configuration

If you want AONS to provide notifications via email, you will need to configure the details for your mail server.

Number	Step	Screenshot
1	Go to the configuration side bar	
2	Click on the "Edit" configuration for the Mail Settings	
3	Choose "No" if you want to enable	

	the mail server. Click Next.	
4	Enter the mail server and listening port.	
5	Enter whether or not this proxy requires authentication. Click Next. If your proxy does not require authentication, skip to step 7.	
6	If your mail server does require authentication, enter the following details: - SMTP Username - SMTP Password Click “Next”	
7	Review and accept changes or go back and change your answers.	

Plugging in to the Local Environment

Initially, on a new AONS instance, there are a fair few system administration tasks to get the instance integrated into the environment. I kind of thing of AONS a bit like an Octopus – you dunk it in your new environment at it starts reaching out it’s arms to the environment around it. This metaphorical “reaching out” is accomplished by connecting to both external registries of format information in addition to connecting to Repositories within your organisation (or ones exposed to you).

Common Tasks

Creating Registry Connections

Number	Step	Screenshot
1	Go to registries tab	
2	Click on “Create” button	
3	Choose type (LC-DFW and PRONOM currently implemented)	
4	Enter Name and URL: Currently http://www.digitalpreservation.gov/formats for Library of Congress (USA) Registry Adapter and http://www.nationalarchives.gov.uk for National Archives (UK) Registry Adapter	
5	Choose Manual or Automatic for Scheduled Updates	
6	If Automatic, continue, if manual, your registry connection should be saved.	
7	Choose the schedule type	
8	Enter details for type specific schedule details (see <<Schedule types and details>>)	

Updating a Registry Connection

Number	Step	Screenshot
1	Go to registries tab	
2	Choose registry to update	
3	Now in the details for the target registry, click on the “Update” button.	
4	Modify details as necessary.	

Deleting a Registry Connection

Number	Step	Screenshot
1	Go to registries tab	
2	Choose registry to delete	
3	Now in the details page for the target registry, click on the “Delete” button.	
4	Choose “Confirm” to really delete the registry.	

Cancelling a Registry Synchronize

Number	Step	Screenshot
1	Go to registries tab	
2	Choose registry with running job (as identified by the "Running"status)	
3	Click the cancel button.	

Creating a Job Schedule on an Existing Registry

When you have a registry with a "manual" schedule set, you can create a schedule the following way.

Number	Step	Screenshot
1	Go to registries tab	
2	Choose registry you wish to add Schedule	
3	If the registry is not idle, cancel the job	
4	Click "Create Schedule"	
5	Choose the schedule's type	
6	Fill in the schedule details as per the type specific parameters, specified <<INSERT LINK TO SCHEDULE PARAMETER APPENDIX>>	

Creating Repository Connections

Number	Step	Screenshot
1	Go to repositories tab	
2	Click on "Create" button	
3	Choose type	
4	Enter in type specific details and a unique name (see <<Repository types and details>>).	
5	Choose Manual or Automatic for Scheduled Updates	
6	If Automatic, continue, if manual, your registry connection should be saved.	
7	Choose the schedule type	
8	Enter details for type specific schedule details (see <<Schedule types and details>>)	

Domain Objects

AONS has a few domain objects worth understanding, both from the aspect of data entry but also when thinking about the relationships between them.

AONS makes usage of hierarchies of object types, which utilises inheritance to add parameters not found on a parent type. When discussing each object, we will ensure to display the hierarchy between each of the objects.

Repository Connections

A repository represents a connection to an existing repository within your organisation, or one which you have been tasked with oversight. For the purposes of AONS, creating/updating/deleting repositories within AONS is about only doing so to the connection – we do not actually ever change the underlying repository; we are interested in reports, not administration.

AONS is interested in the aggregate contents of a repository, not the individual files. We could move down one level of granularity to the individual file, but for now we'll try stay above that and focus on what each collection contains. How do we get this information? We utilise a crawl algorithm – though as we will discuss, this crawl can happen outside of AONS. The crawl algorithm should perform the following high level algorithm:

1. Either manually or as scheduled, perform a crawl on a repository connection
2. Load the connection details for that repository
3. Find the appropriate handler for the repository
4. Request a crawl for the repository
5. The handler should then:
    a. Process each collection recursively
        i. For each file within each collection, create the file's metadata finger print (the unique combination of format name, extension, mimetype etc) which can be used to link it to a Format. At this step, we can optionally employ tools like DROID or JHove.
        ii. For each finger print found, add it to the total aggregate amount
        iii. For each sub-collection, process as a normal collection but add the quantities of finger prints found to the files found in this collection.

There is a repository base type, but the sub-types are where we put most of the details, allowing us to put in quite diverse configuration details in order to connect to them. Unlike registry connections which are detailed below, there is a wide variety of parameters depending on the type of repository specified. This really just gets back to the conflict between a simple API and a fast API. Most of these repositories have web service based API's… but often there is a conflict between getting as little information as possible (to ensure fast scans) and simpler API's. For example, compare the two algorithms for crawling (using file-by-file analysis remotely and locally):

Remotely via web services:

1. Request metadata about the file under scruitiny
2. Process with DROID/JHove
    a. request the entire file byte stream via web services (very expensive)
    b. DROID/JHove quickly identify the file from the first 30 characters
3. Repeat for next file in repository, until all processed.

Locally via low level APIs:

4. Request metadata about the file under scruitiny
5. Process with DROID/JHove
    a. DROID/JHove quickly identify the file from the first 30 characters
6. Repeat for next file in repository, until all processed.

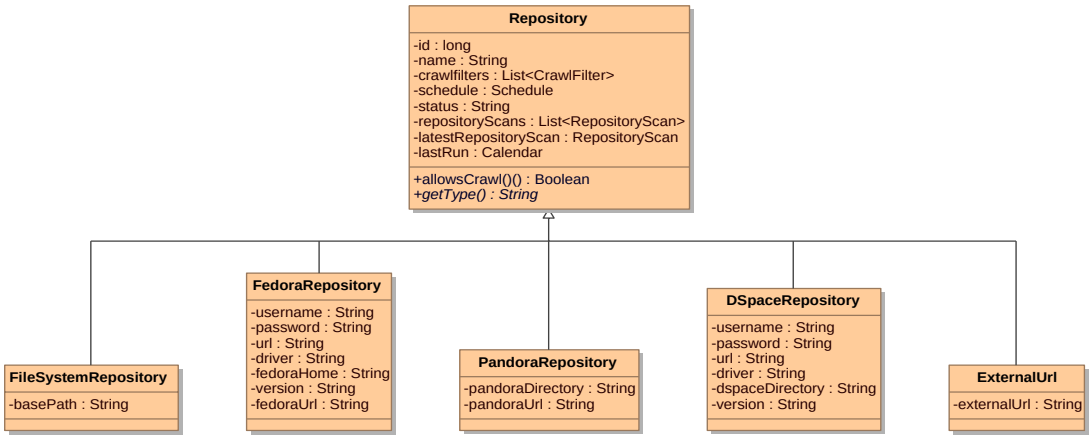
The difference between these two algorithms may seem trivial, but requesting the entire contents of the file when we only need a small part will slow down the algorithm significantly.

There are another two potential features we could and should add to the algorithm: usage of individual file-last-modified metadata. This would be relatively trivial to implement – we'd just have to access within the crawl context. The other possible improvement would be to implement a recovery mechanism should the crawl be interrupted half way through. This type of algorithm isn't as easy to implement since it requires many smaller transactional commits, followed by one at the end to signify the end of the operation. It can be done, but it isn't as easy as you'd think – we already have some quite confusing transaction boundaries for the scheduled work occurring within AONS… this would be yet another bit of complexity.

Supported Repository Types

AONS currently has support for the repository types listed here:

- File System
- DSpace
- Fedora
- External URL
- Pandora (and repository system used only within the NLA)



We also have planned support for the following repository types:

- Trim
- E-Prints
- SRB (via an effort going on from Jane Hunter at the University of Queensland)

Implementing a new repository type is relatively easy – probably the hardest part about implementing it is actually getting it into the GUI. The actual interface backend handlers should be relatively easy to familiarise yourself with, as long as you have a firm grip on the target repositories API. Also, from the creator of a repositories perspective, it is often easier to implement the XML summary required by the External URL adapter then to create your own handler.

Base Repository Attributes

The base repository has a number of attributes which are found on all repositories. Most of these are not directly input but created when repository crawls are performed. The table below details each attribute, it's type, description and origin:

Attribute Name	Type	Description	Origin
id	Long	Unique identifier for this repository connection.	Generated On Creation
name	String	Unique name for this repository connection.	Entered on creation/update
crawlFilters	List of CrawlFilters	Crawl filters showing which files to exclude	Should be entered via GUI, currently needs to be implemented
schedule	Schedule	Crawl Schedule for this repository, if null the repository is implicitly manual only in operation. See the schedule attributes for details on the required parameters	Entered on creation and also specifically by create/update/delete from the Repository details page.
status	String	Indicator of this	Application controlled: all

		repositories status (Idle|Running|Cancelling)	repositories are implicitly Idle upon startup and only changed when either manually invoked or started as a part of a schedule.
repositoryScans	List of RepositoryScan	For every repository scan we perform, we also generate one of these objects.	Generated as part of a crawl
latestRepositoryScan	RepositoryScan	This is a convenience object which holds the latest repository scan performed.	Generated as part of a crawl
lastRun	Calendar	The last run performed	Generated as part of a crawl

FileSystemRepository Attributes

Attribute Name	Type	Description	Example
basePath	String	Must be a valid folder on the file system readable by the AONS server process.	/dev/workspace Windows: file:///C:/dev/workspace *nix: file:/dev/workspace

FedoraRepository Attributes

Attribute Name	Type	Description	Example
username	String	Database username	fedora
password	String	Database password	s5w0rdpA
url	String	Database URL connection String	jdbc:postgresql://localhost:5432/fedora
driver	String	Database driver for JDBC connection	org.postgresql.Driver
fedoraHome	String	This is the location on disk of the fedora home directory	/opt/fedora Windows: file:///C:/opt/fedora *nix: file:/opt/fedora
version	String	Version of the fedora	*<<Not currently accessible via GUI>>*

		installation – currently irrelevant since we only support version 2.2	

PandoraRepository Attributes

The Pandora repository makes usage of a beta Ruby service to identify collection ids. This was seen as preferable to having it access the backed database tables (but we always could change it if the Ruby service does not make it past beta).

Attribute Name	Type	Description	Example
pandoraDirectory	String	Base directory of the Pandora installation to scan	/pandas/prod Windows: file:///C:/pandas/prod *nix: file:/pandas/prod
pandoraUrl	String	URL for the Ruby collection service	http://www-devel.nla.gov.au/rbpandora/browse/tepfeed

DSpaceRepository Attributes

Attribute Name	Type	Description	Example
username	String	Database username	dspace
password	String	Database password	s5w0rdpA
url	String	Database URL connection String	jdbc:postgresql://localhost:5432/dspace
driver	String	Database driver for JDBC connection	org.postgresql.Driver

ExternalUrlRepository Attributes

Please note that the ExternalURL type is expecting an XML structure which as of yet does not have a locked down schema. For an example of it as it currently exists, please see the Example Format Summary XML file listed in the Appendix.

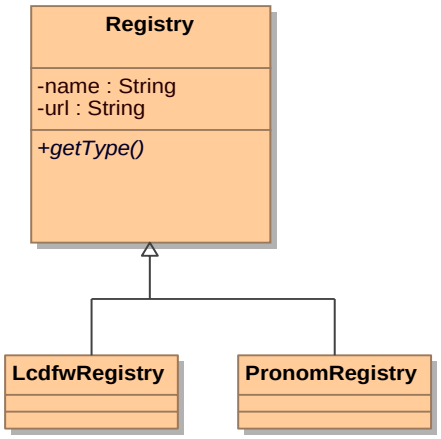
Attribute Name	Type	Description	Example
externalUrl	String	Location of the service providing	http://localhost:8080/aons2/ExampleFormatScan.xml

		the format scan XML file.	

Registries Connections

Description

Registries connections represent the connections to remote repositories of information. There is a base registry type which is extended by each implementation. Currently we have one for Library of Congress' (USA) Digital Format Website (LC-DFW) and one for the National Archive's (UK) web-based technical registry (PRONOM). We had also planned to support the Global Digital Format Registry (GDFR) but were unable due to development time constraints.



Why do we use registries? Well, registries contain information on formats – they are, in theory, the most up to date source of information relevant to a format and it's obsolescence. Our intended goal with registries is to use their quite rich data model intelligently to help us determine which formats are at risk within our local repositories. We have a few barriers to that:

1. The registries are quite diverse in their data structures
2. The registries do not contain all file formats – obscure "at risk" formats are not often mentioned
3. The registries do not contain a single value we can use to look for a risk metric – the data contained is subjective rather then quantitative.

The last point is probably the most crucial: until we have a quantitative measure of a formats risk, however that is generated, the registries really only provide enough information to help a human decide a format's risk. We did investigate using what data was present in a questionnaire style format to auto calculate a formats risk… but found that this was very complicated, especially considering that much of the information was not present (an appropriate analogy would be trying to create an algorithm to rate how good a movie is – something which at the end of the day, has to come from the 'gut'). So, instead we perform a "synchronize" with external registries,

taking down all of their subjective data, creating a set of RegistryFormats. With this data available locally (cached) we can then link this data to format metadata fingerprints found within a repository.

So, to summarise, a registry connection within AONS represents a connection to an external source of subjective information relevant to the assessment of risk on a Format. Later, should external registries provide quantitative information regarding the risk of a format, we will aim to use that in preference to asking a user to judge a formats risk. We've discussed a few good possible ways to get this information – either by extending GDFR or PRONOM, but within this iteration of the project it is not looking likely to happen. Ideally, we'd like a system in place where an AONS installation can perform risk assessments which are then fed upstream to a service (GDFR or PRONM) which is then uses an active voting community judge which risk assessments are valid and which are out of place. Still, this idea is currently just that, an idea, but hopefully will be implemented with the next iteration of AONS.

Registry Connection Parameters

Unlike repository connections, each registry so far has required only two parameters upon configuration: A unique name and a URL.

Parameter Name	Type	Description
id	number	The id of the registry
name	text string	This is the logical name of the registry.
type	text string (read-only)	This defines the type of the registry – it is used to find the appropriate handler when we perform our synchronize operation.
url	text string	This is the base URL which we use to contact the Registry. It is context sensitive to the type of registry: Library of Congress (USA) is http://www.digitalpreservation.gov/formats and the National Archives (UK) is: http://www.nationalarchives.gov.uk.

Formats

The AONS application deals with the risk associated with formats within a repository, and in combination with the quantities of the formats found in the configured repository connections, deals with the risk in a given repository. AONS has two concepts of Formats:

1. External Formats, which are cached from format registries.
2. Internal Formats, which we apply risk assessments to.

As mentioned before, should an external format provide a risk assessment, we remove the need to have internal formats which we apply risk assessments to. However, until we see risk assessments being performed by an external service, we need the internal format object.

Base Format Attributes

The base format has the following attributes.

Attribute Name	Type	Description	Example
id	number	The id of the format. Auto-generated when the format is created.	1
name	String	The name of the format	“Graphical Interchange Format”
version	String	The name of the format’s version	“1.0”
lastUpdated	Calendar	The date of the last update of this format. Not modifiable via human, but readable via REST	“1997-07-16T19:20:30.45+01:00”

AONS Format Attributes

The AONS format represents a managed object within the system. It is on this object which we apply risk workflows. An AONS format extends the base format and has the following additional properties:

Attribute Name	Type	Description	Example
id	number	The id of the format. Auto-generated when the format is created.	1
nameAliases	list of NameAlias	Name alias sub-entity objects which allow this format to have multiple known name	*See name and version on base format and also look at REST nested object notation section.*

		version combinations.	
puid	String	PRONOM Identifier	*fmt/000017*
puidAliases	list of PuidAlias	PUID alias sub-entity objects which allow this format to have multiple known PUIDs.	*fmt/000017, fmt/000018*

Registry Format

Schedule

Format

Format Metadata Fingerprints

REST Interface

REST Operation Conventions

REST URL Syntax

REST resources are queried via the following syntax:
${baseApplicationUrl}/rest/${restResourceName}

So as an example for an AONS deployment on local host and wanting to access all registries (via the virtual query REST object Registries):
http://localhost:8080/aons/rest/Registries

Method Overriding

Typically, REST uses the HTTP operations PUT, GET, POST and DELETE to map to meanings Create, Retrieve, Update and Delete respectively. This is fine and dandy when your code is running since it should be able to manipulate the HTTP operation via the API, but for testing, this approach is very limiting. So, to get around this, we put in place a “methodType” parameter, which if added to a GET url can be used to change the operation to one of the other types. What this means, is that you can do simple testing of REST urls via a browser. Neat, huh.

Create Operations

To create a REST resource, you need to remember that you are performing a HTTP PUT operation. This PUT operation should *not* include the identifier since, well, there isn't one yet. This identifier is only created when the object is saved into the system.

For example, if we wanted to create a new Library of Congress registry, we would perform a HTTP PUT with the following URL and attributes:
URL: http://localhost:8080/aons/rest/Registry
Attributes:

name=Test Registry
url=http://www.digitalpreservation.gov/formats
type=LC-DFW

You'll notice we needed to specify the type of the registry to demonstrate which actual connection handler we'll use to process this registry. These values are detailed below for each REST resource with multiple implementations.

Now, I hear you ask, if we don't give it an identifier when we create it, how do we know what to retrieve for a query? Well part of the creation of the resource is to generate this identifier. This identifier is then returned back after the REST call has completed in a successful response. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<success-response>
    <domain-object-type>Registry</domain-object-type>
    <method-type>PUT</method-type>
    <id>182</id>
</success-response>
```

Retrieve Operations

When performing a retrieve operation, you will need to know the REST resources identifier (id). So for example, to perform a retrieve on a registry with Id 10, you would perform a HTTP GET with the following URL:
http://localhost:8080/aons/rest/Registry?id=182

This would return the following output:

```
<?xml version="1.0" encoding="UTF-8"?>
<registry>
    <id>182</id>
    <name>Test</name>
    <url>http://www.digitalpreservation.gov/formats</url>
    <type>LC-DFW</type>
    <latest-statistic-historical>
        <id>183</id>
        <change-amount>0</change-amount>
        <character-count>0</character-count>
        <number-added>0</number-added>
        <number-deleted>0</number-deleted>
        <number-of-formats>0</number-of-formats>
        <validFrom>2007-10-23T11:44:05.351+10:00</validFrom>
    </latest-statistic-historical>
    <latest-usage-historical>
        <id>184</id>
```

```
        <number-utilised>0</number-utilised>
        <validFrom>2007-10-23T11:44:05.351+10:00</validFrom>
    </latest-usage-historical>
    <statistic-historicals>
        <id>183</id>
        <change-amount>0</change-amount>
        <character-count>0</character-count>
        <number-added>0</number-added>
        <number-deleted>0</number-deleted>
        <number-of-formats>0</number-of-formats>
        <validFrom>2007-10-23T11:44:05.351+10:00</validFrom>
    </statistic-historicals>
    <usage-historicals>
        <id>184</id>
        <number-utilised>0</number-utilised>
        <validFrom>2007-10-23T11:44:05.351+10:00</validFrom>
    </usage-historicals>
    <status>Idle</status>
</registry>
```

Update Operations

Updates are based on a create operation but have one fundamental difference, they require the identifier and don't return the identifier after the operation is completed. One point to note with the updates is that the update will only change the attributes specified; if you omit an attribute, the bind operation will not over write all other attributes with null. This means that your update URLs can be quite concise.

So, continuing the example from before, if we wanted to update the name of the registry, we'd perform the following operation via a HTTP POST:
URL: http://localhost:8080/aons/rest/Registry
Attributes:

- id=182
- name=Changed Name

You should see a response message like so:

```
<?xml version="1.0" encoding="UTF-8"?>
<success-response>
    <domain-object-type>Registry</domain-object-type>
    <method-type>POST</method-type>
</success-response>
```

Delete Operations

This is the simplest part of a REST call, and these are just the combination of a HTTP DELETE call along with the URL and an attribute with the identifier.

So, to delete the registry we've used as an example up until now, we perform this operation:
URL: http://localhost:8080/aons/rest/Registry
Attributes:

- id=182

You should see a response like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<success-response>
    <domain-object-type>Registry</domain-object-type>
    <method-type>DELETE</method-type>
</success-response>
```

Search Operations

Often, you will want to search for all of a type of domain object. For example, you may want to list all registries in the system. To do this, we use a virtual domain object – not all REST resources have them, but they will be specified in the REST Domain Object section if they do. So, to list all registries, you could put in this HTTP GET URL:
http://localhost:8080/aons/rest/Registries

Sometimes, the volume of these domain objects is so large you will want to also include a filter. For example, you may wish to filter formats based on a search string:
http://localhost:8080/aons/rest/Formats?searchString=JPEG

It is important to note that the structure of the result returned from a search group is often only a subset of the attributes available when performing a normal retrieve. It should be viewed that a search is a summary of domain objects whilst a retrieve with an identifier returns the complete view.

The search may also return a relevance ranking for the search results if a search filter was used.

REST Error Messages

We have tried to be relatively verbose with the error messages returned from AONS to ensure that calling code can handle them well. To do this, we have adopted the best practice of:

- Sending back a HTTP error response of 500 on errors
- A standard template for errors including:
    - A code
    - An error status value
    - A detailed message

Here is a typical error:

```
<?xml version="1.0" encoding="UTF-8"?>
<failure-response>
    <code>GeneralException</code>
    <message>Can not perform method [PUT] on [Format], available
methods [GET]</message>
</failure-response>
```

For the following types of errors, we also have additional attributes available:

- Invalid parameter (also includes the invalid parameter and it's value)
- Missing parameter (also includes the missing parameter)

REST Domain Objects

AONS set out from the beginning to allow usage via REST interfaces. Almost all domain objects can be created, retrieved, updated and deleted directly via simple REST methods. This section covers over the various REST "resources", including what AONS object they map to and what REST methods they allow out of the create, retrieve, update and delete stack. Further we also discuss searching, since some domain objects allow querying via a virtual REST object.

AONS Resource Name	Registry
REST Resource	Registry
Available CRUD Methods	All
Searching Virtual Resource	Registries
Notes	Also requires an extra attribute called "type" which dictates what entity type the Registry is (National Archives (UK) or Library of Congress(USA)). These types are either PRONOM or LC-DFW respectively.

AONS Resource Name	Repository
REST Resource	Repository
Available CRUD Methods	All
Searching Virtual Resource	Repositories
Notes	Also requires an extra attribute called "type" which dictates what entity type the Registry is (National Archives (UK) or Library of Congress(USA))

AONS Resource Name	Format
REST Resource	Format
Available CRUD Methods	Retrieve Only
Searching Virtual Resource	Formats
Notes	In addition to providing explicity access to the two subtypes of BaseFormat (Internal and External Format) we have also provided this REST resource so as to allow people to perform a query on both at the same time.

AONS Resource Name	Internal/AONS Format
REST Resource	AonsFormat
Available CRUD	All

Methods	
Searching Virtual Resource	AonsFormats
Notes	This provides access to all the CRUD operations on internal formats.

AONS Resource Name	Log Message
REST Resource	LogMessage
Available CRUD Methods	Retrieve Only
Searching Virtual Resource	LogMessages
Notes	Allows access to the log messages within AONS. Can search upon the following fields: - logLevel (must be one of “Debug”, “Information”, “Warning”, “Error” or “Fatal”) -