ConservationSpace: Interface Control Document

Integration with Partner CMS and DAMs

# Introduction

An important feature of ConservationSpace is its ability to import cultural object data and imagery from partner CMS and DAM systems located behind a network firewall. This document describes infrastructure and services required by the ConservationSpace application and defines the interface for such services. In order to establish integration between ConservationSpace and partner CMS and DAM systems, each partner must provide an implementation of the services described in this document.

# CMS and DAM Services Required by ConservationSpace

As outlined in the CMS and DAM integration design document, the ConservationSpace system must be able to perform the following operations with partner CMS and DAM systems:

* Search for cultural objects within one or more partner CMS systems using a limited number of search criteria such as title and artist name
* Acquire thumbnail images of cultural objects from one more partner DAMs for the purpose of rendering search results
* Acquire cultural object data and object relationships, e.g. parent / child and images related to the cultural object
* Detect cultural objects that have been updated recently so that ConservationSpace can keep its cultural object library in sync with partner CMS systems
* Search for and acquire full size images in a variety of formats from partner DAMs and allow expansion of the API to include other types of media in the future
* Send error messages to the partner system when processing exceptions occur

# Integration Approach

## Network Connectivity

ConservationSpace will, by necessity, be the initiating party in all CMS and DAM interactions. However, partner CMS and DAM systems are typically not located on a public network. Therefore, partner institutions will need to coordinate with the ConservationSpace hosting provider to establish an acceptable network path that exposes the partner’s interface implementations to the ConservationSpace system. Some of the typical options include:

* Persistent VPN\*
* SSH tunneling aided by network appliances or tools such as autossh\*
* Opening holes in the partner firewall to permit communications from the ConservationSpace system to reach the partner service endpoints
* Hosting the partner service end points in a DMZ

\*The NGA has elected to use an SSH tunnel for development and testing and a DMZ accessible only to ConservationSpace for production. .

## Protocol and Formats

Sirma wanted to keep the interface implementation as simple as possible for partners to implement so JSON over the HTTP protocol was selected as the primary format and protocol for the CMS and DAM integration services. The service end points and URL structures were designed with the simplicity of REST in mind although they do not seek to provide a full RESTful implementation.



Figure 1 - Sequence diagram of CMS/DAMS Integration

## Required Service Implementations

All partners providing an integration implementation shall supply a web service endpoint (a URL) for each of the six required services below. The URL listed below each service is a suggested naming convention and will be what the NGA uses. However, each partner is free to choose any URL prefix they wish. The ***mandatory portion*** of URLs is highlighted in ***bold italics*** in the following table.

|  |  |
| --- | --- |
| **Service** | **Description** |
| 1 - Retrieve Cultural Object | Web service that, when appending a source system identifier and an object ID to the URL with a .json extension and using the HTTP GET method, returns the corresponding full cultural object record.  https://<host>/art/<***source>/objects/<object id>.json***  *Purpose: required in order to import cultural object records into ConservationSpace* |
| 2 - Search for Cultural Object Records | Returns an abridged cultural object record, a service URL, and thumbnail (if available) for each cultural object in the search result. If source is omitted, all CMS sources should be searched.  https://<host>/art/<source>/***objects.json?<search parameters>***  *Purpose: to located art objects within a CMS for the purpose of importing them into ConservationSpace*  Note: ConservationSpace will always populate a query string regardless of the METHOD used, e.g. always populate the pagination values even when the POST method is used, in order to distinguish a read-only search operation from potential future write operations. |
| 3 – Retrieve Media content by ID | Web service that accepts a unique media ID and returns the binary media content itself, either via an HTTP redirect to the source of the media on a different web server, if it is known to be accessible directly by the ConservationSpace system, or a proxied stream of the original source directly to the caller. The mime type of the response should be set to the proper media mime type, e.g. image/tiff. ConservationSpace will validate the format of the file after the download is complete.  https://<host>/media*/<****source>/<media-type>/<media id>***  *Purpose: for acquiring media content from partner DAM systems* |
| 4 – Retrieve Media Record by ID | Web service with a .json extension that accepts a unique media ID and returns the metadata record for that media. .  https://<host>/media/***<source>/<media-type>/<media id>.json***  *Purpose: for acquiring media records from partner DAM systems for the purpose of importing that data into ConservationSpace* |
| 5 - Search for Media Records | Returns abridged media records similar to what Service #2 provides for cultural objects. The result set may include a thumbnail of each media item in the result set. If source is omitted, all DAM sources should be searched. Media-type is a required field and for now, only “images” needs to be supported.  https://<host>/media/<source>/<media-type>***.json?<search parameters>***  Purpose: for locating media to be included in ConservationSpace documents  Note: ConservationSpace will always populate a query string regardless of the METHOD used, e.g. always populate the pagination values even when the POST method is used, in order to distinguish a read-only search operation from potential future write operations. |
| 6 – Remote Log Service | Web service that accepts a POST request from ConservationSpace and merely logs the posted contents as a message so that it can be viewed later for diagnosis of problems with the ConservationSpace interface.  https://<host>/system/logger***.json***  *Purpose: facilitates debugging and operations by capturing detailed responses relevant to this interface from the ConservationSpace system* |

## Responsibilities

### ConservationSpace Responsibilities

* Initiates and coordinates the requests for search, import, update, etc.
* Holds the registered integration subsystems and their configurations (see Partner Configuration Options section below)
* Transformation of data takes place before sending a request and after receiving a response. Data transformation facility takes care of mapping received fields to ConservationSpace partner ontology
* Sends error messages to the error logger service when there are problems with the data received in response to a request

### Partner Responsibilities

* Provide http/s communication endpoints to be accessible by ConservationSpace through a network channel to be coordinated with the hosting provider.
* Read and process requests and respond appropriately as outlined in this interface control document
* Optionally read and authorize the supplied authorization token when receiving requests over HTTPS protocol
* Optionally strip all formatting prior to submission of data since data received by ConservationSpace through the interface will be saved “as-is”. Embedded HTML and other formatting will not be displayed in the ConservationSpace application – the formatting codes will be displayed as literal text.
* Maintain the partner-specific code lists used in the partner ontologies defined within the ConservationSpace system. When a required property that is mapped to a ConservationSpace code list cannot be matched on import through the interface, ConservationSpace will throw an error. For optional properties, a warning will be issued and the value will be discarded.

## Benefits of the Proposed Approach

* Easy: Simple to understand and easy to implement
* Sustainable: Widespread usage; support across many different programming languages and system types; plenty of programming examples to draw upon and frameworks to speed development
* Flexible: type-less JSON responses are flexible enough to accommodate all partner cultural object ontologies and are human readable when formatted with indentations
* Fast: GET requests present an opportunity for caching results to the file system, even for searches

Reuse: JSON formatted responses are system-agnostic meaning they could be reused for other purposes once implemented. ConservationSpace will simply ignore any data elements it does not recognize.

# Interface Specifications

## Partner Configuration Options

Each tenant / partner in the system will have the following integration parameters available for definition in the ConservationSpace tenant administration configuration:

* HTTP client timeout – maximum time for ConservationSpace to wait for the start of a response from any of the service end points – once a response has started, ConservationSpace will wait as long as required for the entire response to be transmitted.
* Separate URL prefixes for each of the eight service end points
* Authorization security token (a string) that will be sent by ConservationSpace with any HTTPS request when making a service request
* ConservationSpace R2 settings (with example values – this list will be updated as additional setting attribute names become available):
  + eia.cms.communication.uri=http://localhost:12000/cms\_banchev.bg
  + eia.cms.communication.services={"search":{"GET":"service/art/objects.json"},"retrieve":{"GET":"service/art/objects/{0}.json"},"logging":{"POST":"service/log"}}
  + eia.cms.communication.security.token=eyJhbGciOiIxMTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ
  + there will also be services for media / image searches

a client timeout setting will be embedded with each of the above options

## Common Security Measures

* The ConservationSpace system will initiate all network requests associated with the integration. As previously mentioned, the connection between ConservationSpace and the tenant must be established via a series of conversations between the hosting provider and tenant’s network and system engineers.
* The HTTP authorization header (<https://en.wikipedia.org/wiki/Basic_access_authentication>) will be sent with the request by ConservationSpace if a security token value is configured with the service end point being used. The partner implementation may then validate this token to authorize the caller.
* HTTP as well as HTTPS may be specified for each of the service URLs. If the https protocol is to be used exclusively, then requests to services over HTTP should either be ignored or redirect to the proper HTTPS service. ConservationSpace will only send the authorization header when HTTPS is in use.
* By default, ConservationSpace is configured to validate TLS certificates, but if partners are using self-signed certificates, there is a partner specific configuration option for disabling certificate checks. If a commercially signed certificate is used and ConservationSpace is configured to verify TLS certificates, the root certificate authority signing the certificate must exist in Java’s default list of root CA certificates. In addition, any intermediate certificates comprising the complete chain of trust should be configured in the partner’s web server, otherwise the verification test will fail. The NGA has a simple Java based tool available for any partners wishing to verify that their TLS certificate chain can be verified.

## Response Formats and Structures

JSON will be the format used for all text responses. Values in requests and responses should represent the natural data type for the field being represented, e.g. numerical, Boolean, string, date. Some fields are mandatory for all partners regardless of the ontology in play, while other fields are mandatory based on the partner’s ontology definition. Media binary content is the only non-JSON response and will be transmitted as binary data with the mime-type set to the mime-type corresponding to the media asset’s type and format.

The source system of record will be supplied with each entity as well as in the URL of the cultural object and media record services. The source name should be reasonably short and should use characters allowed in URLs without having to escape them. The NGA will initially use one source for cultural object records (“tms”) and two sources for media (“web-images-repository” and “portfolio\_dclpa”) with a third media source (“edam”) in the future. The source is validated by ConservationSpace against a code list, so each partner should identify the various sources for CMS and media to Sirma so they can be entered into the ConservationSpace code list configuration.

The namespace of record properties will be specified in order to convey the type of entity contained in the response. For example, the namespace “cultObj” will be used for cultural objects and the namespace “image” will be used for images. In future releases, there could be additional media namespaces including a generic “media” namespace, but for now only “images” will be used since that is the only type of media search and record import supported by ConservationSpace R2. The namespace will help to identify the type of entity being returned as well as to differentiate between properties with the same name in multiple entities, e.g. image:id vs. cultObj:id. Search queries will include the namespace of the property in the request parameter if it is different from the default namespace associated with the service, e.g. cultObj:id rather than id which might otherwise be interpreted as image:id. Note that ConservationSpace R2 will always include the namespace when providing property names to partner integration services.

## Namespaces

|  |  |
| --- | --- |
| **Namespace** | **Description** |
| cultObj | Cultural Object namespace – this namespace encapsulates properties that are associated with cultural objects |
| image | Image namespace – this namespace encapsulates properties that are associated with images |

## JSON Response Formats

### Result Set: Records

The search results JSON construct includes an array of items and paging controls. Each item consists of a URL, a collection of record properties and references, and an optional thumbnail image depicting the item if a suitable thumbnail exists to represent the item.

items : array of result structures

url : the service end point where the item record can be found

record : record structure for either a Cultural Object or Media asset

thumbnail (optional but encouraged) : URL to thumbnail or base64 encoded binary image data

paging : collection of properties

skip : skip over this many items before spooling results

limit : number of items to return in this page of results

total : total number of items that result from the query across all pages

#### Implementation Notes

* url – the supplied URL should be the URL to the record for that item, for example, if the item is a cultural object, the URL should point to the fully qualified Service 1: Cultural Object by ID service URL for this cultural object, e.g. https://data.nga.gov/art/tms/objects/1138.json
* paging – the paging controls in the response indicate the total number of results and, via the skip and limit controls, the relative location of this set of results within the entire result set. skip, limit, and total should be provided in requests and responses as numerical values. A maximum limit value of 1000 should be enforced to avoid excessively long responses. However, the total number of search results must not be limited or altered and should reflect the size of the entire result set.
* multi-line strings – at times, it will be necessary to return concatenated values in a single field or to return large texts in paragraph form, e.g. bibliography entries or provenance text A single newline character (char) 10 should be used to represent a return + line feed in your data. ConservationSpace will then make any necessary transformations based on the widget in which this multi-line string data is being displayed.
* thumbnail - each item in the result set may have a thumbnail associated with it. Thumbnails can be defined as either a base64 encoded binary of the actual thumbnail image which is sent in-line with the response or as a URL to the actual image which is accessible to all users of the ConservationSpace system for this tenant. The design of ConservationSpace allows for a bounding box of 100x100 pixels for thumbnails of cultural objects and DAM images so it is optimal to supply a thumbnail with a 100 pixel maximum extent if possible. If any other size is supplied, the design may cause the web browser to resize the image to fit the allotted space. *Caution: If thumbnails smaller than 100 pixels on a side are supplied, while the application will not alter the aspect ratio of the thumbnails, the design might cause the web browser to upscale the image to fit the space and therefore might cause thumbnails to appear pixelated. It is better to provide a slightly larger thumbnail than a smaller one in the event that the exact size is unavailable.*

### Record: Cultural Object

The properties of a cultural object will vary widely among the partners as they all have different ontologies. The default namespace of the cultural object record shall be “cultObj” and this value shall be defined in the namespace property of the record. Properties from other namespaces may be included prefacing the property name with the alternative namespace, e.g. “image:id”. The properties for cultural objects that are common to all partners are underlined below. **Mandatory properties are in bold** and properties that are *mandatory only if they exist are embellished in italics*. All other properties are optional. There is a mapping defined for each partner between the ConservationSpace cultural object ontologies and properties in the CMS and DAM interfaces.

record : collection of cultural object properties and references

**namespace** : cultObj

**source :** short name of the system of record for this object, e.g. “tms”, “collectionspace”, etc.

**id**: partner specific unique id for this object within the given source

**classification** : drawing, painting, sculpture, etc. required for property ontology mapping

**title** : primary title of this cultural object

**artistNames :** name(s) of the artists who created work –NGA mandatory

**accessionNum :** NGA mandatory – accession number for the object

description : NGA optional, single valued, property

dexid : NGA optional, single valued, property

***references*** : associations between this cultural object and other entities (cultObj, images)

{ **predicate** : hasChild | hasParent | hasSibling

object **: abridged cultural object record, i.e. a** collection of **common mandatory properties** and properties required to render the partner’s cultural object search results header for this entity

**namespace** : cultObj

**source** : short name of system of record

**id** : partner specific unique id for this cultural object

**classification** : drawing, painting, sculpture, etc.

**title** : primary title of the art object

**artistNames** : name(s) of the artists who created work

**accessionNum :** NGA accession number of the object

}

{ **predicate** : hasPrimaryDepiction | hasDepiction

object : abridged image record, i.e. a collection of **common mandatory properties** and properties required to render the image search results header

**namespace** : image

**source :** short name of the system of record

**id** : when used with hasPrimaryDepiction, the partner defined ID of the primary image depicting this object should be used and it should represent the largest image available that constitutes a “general” and full view of the cultural object in a natural way, e.g. visible light; if multiple primary depictions are provided, only the first one will be automatically imported as the dossier image within ConservationSpace

**classification** : type of image – used to map to partner ontology, e.g. “image”, “photography”, “scientific\_image”, etc.

**fingerprint** : a partner computed value supplied for change detection

}, …

An unabridged cultural object record for a partner would include all properties for the particular classification of the object and, for the NGA, a fully formatted JSON response might look like this:

{

"record": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods",

"artistNames": "Bellini, Giovanni (artist); Titian (artist)",

"attribution": "Giovanni Bellini and Titian",

"accessionNum": "1942.9.1",

"bibliography": ["Hartshorne, Rev.C.H.A Guide to Alnwick Castle.London, 1865: 62.", "<i> Paintings in the Collection of Joseph Widener at Lynnewood Hall. < /i> Intro. by Wilhelm R. Valentiner. Elkins Park, Pennsylvania, 1923: unpaginated, repro., as by Giovanni Bellini."],

"displayDate": "1514 / 1529",

"creditLine": "Widener Collection",

"description": "The Feast of the Gods, Giovanni Bellini and Titian, 1514 / 1529",

"location": ["1942.9.1 – West Building Main Floor Gallery 12"],

"homeLocation": ["1942.9.1 – West Building Main Floor Gallery 12"],

"inscription": "lower right on wooden tub: joannes bellinus venetus / p MDXIIII",

"medium": "oil on canvas",

"departmentAbbr": "DCRS",

"dimensions": "overall: 170.2 x 188 cm(67 x 74 in .) \n framed: 203.8 x 218.4 x 7.6 cm(80 1 / 4 x 86 x 3 in .)",

"provenance": "Probably commissioned by Alfonso I d 'Este, Duke of Ferrara [d. 1534);[1] by inheritance to his son, Ercole II d'Este, Duke of Ferrara[d .1559];by inheritance to his son, Alfonso II d 'Este, Duke of Ferrara [d. 1597]; by inheritance to his cousin, Cesare d' Este...",

"references": [{

"predicate": "hasPrimaryDepiction",

"object": {

"namespace": "image",

"source": "intranet",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef",

"classification": "photography",

"fingerprint":"352e146e8195364ade2c5f34bf1aead29b1fbbe91c",

}

}]

}

}

#### Implementation Notes

* The ontology model (see services at the end of this document) should be consulted before embarking on an implementation as it lists the properties that are considered to be mandatory for each partner, based on the cultural object classification. In other words, for a drawing, there might be a different set of mandatory fields than for a painting. Only the ontology specific to each partner defines these requirements. There is no harm in including all fields regardless of the cultural object’s classification as ConservationSpace will simply ignore extraneous fields.
* any property other than “id”, “namespace”, “source”, or “classification” may be defined as a multivalued property as needed – ConservationSpace will automatically detect both the cardinality and the data type (numerical, Boolean, date, or string) based on the JSON response.
* classification is a mandatory property that is used to map the cultural object to the appropriate ontology model within ConservationSpace – *the partner data model should be consulted in order to determine the appropriate logic to use when computing the classification property*
* references - Cultural objects potentially have relationships with other cultural objects, for example, parent, child, or sibling relationships, or with images. These relationships will be represented in conservation space via a simple “associated with” relationship.
* ConservationSpace will support multi-value properties via JSON arrays. However, nested collections of properties not specifically defined by the interface will be ignored by ConservationSpace. For example, these are ok:

“property” : “value” OR “property” : [“value1”,”value2”]

but this is NOT:

“property” : { “subproperty” : “value” }

* We recommend using a JSON programming library to ensure that the services emit property formatted JSON with values that are properly escaped

### Record: Image

Similar to cultural objects, images also have properties that are common for all partners, some **mandatory**, some *mandatory only if they exist*, and some optional. Partners also have mandatory and optional properties that are based on their ontologies. The default namespace of “image” shall be used for image records and this namespace must be specified as a property of the record. The common set of mandatory properties for all partners for media include: namespace, source, id, and classification.

record : collection of image properties

**namespace** : image

**source** : short name for the system of record for this image

**id** : partner’s unique identifier for this image

**classification** : type of image, e.g. “photography”, “scan” etc. - mapped to partner’s image ontology

**fingerprint** : a partner computed value supplied for this record that is used for change detection

**title** : name or title of the image

**lastModified** : time and date the image was last modified

**lightType** : NGA mandatory property

**orientation** : NGA mandatory property

description : NGA optional property

***references*** : associations between this image and other entities (cultObj, images)

{ **predicate** : primarilyDepicts | depicts

object **:** abridged cultural object record

**namespace** : cultObj

**source** : short name for system of record for cultural object **id** : partner specific unique id for this cultural object

**classification** : drawing, painting, sculpture, etc.

**title** : primary title of the art object

…

},

{ **predicate** : relatedAsset – used to define image-to-image relationships

object **:** abridged image record

**namespace** : cultObj

**source** : short name for system of record for image

**id** : partner specific unique id for this image

**classification** : scientific\_image, photography, generic image, etc.

**fingerprint** : partner computed value supplied for change detection

}, …

Notes:

* Associations with entities such as cultural objects will be recorded as an “associated with” relationship in ConservationSpace R2. While the model allows for the definition of different types of related entities, ConservationSpace R2 will only process relationships defined for cultural objects and other images. Entities encountered in references with unrecognized namespaces will be ignored. Each partner will have an opportunity to map their relationship predicates to a ConservationSpace relationship type.
* image properties supplied with the image record will be mapped to the image ontology on a per partner basis – e.g. “spectrum” might be a partner supplied property for images which is mapped to a specific ConservationSpace ontology property for scientific images called “lightType”
* fingerprint is a partner computed or supplied value that should change whenever the image contents or the image metadata is changed. In many cases, the lastModified date might be reliable enough to use for change detection and in such cases, partners may simply supply this value as the fingerprint. However, if the lastModified date is not reliable or the partner’s DAM system allows for image content to be updated without altering the lastModified date, a separate fingerprint should be computed, e.g. a reasonably long crytographic hash of both the image content and the image metadata. The ConservationSpace system will store the fingerprint supplied with media records and will use it when determining whether there are updates available for a given media record.

An unabridged description of an image record for a partner would include all properties for the particular type of image and, for the NGA, a fully formatted JSON response might look like this (note that some properties in media records, e.g. mimetype, are used only by the NGA search header template in search results rendered by ConservationSpace since ConservationSpace will set the system property mime-type for media items based on the content-type of the service used to fetch the contents of the media item rather than the equivalent property returned by the media record or search services):

{

"record": {

"namespace": "image",

"source": "intranet",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef",

"classification": "image",

"mimetype": "image/tiff",

"title": "60-primary-0-nativeres.ptif",

"lastModified": "20160421T182434-04:00",

"fingerprint": "3537aac2467dce2e146e8195362c5f34bf1aead29b1fbbe91c",

"pixelsPerCM":489.34325, "lightType": "Normal Light",

"orientation": "Front",

"description": "after treatment of 1942.1.9, normal light",

"references": [{

"predicate": "primarilyDepicts",

"object": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods"

}

}]

}

}

## Service 1: Retrieve Cultural Object

Retrieves the cultural object definition for the supplied cultural object and returns the full, unabridged cultural object record as defined in *Record: Cultural Object*. This service is used by ConservationSpace when importing or updating a cultural object. Whenever a cultural object is imported or updated, the primary object image(s) are also checked to see whether they require updating. For this reason, partners may optionally implement a feature whereby the object record modification date is updated when its primary image(s) are updated so that the ConservationSpace automatic CMS update process detects the new image(s).

### Request

HTTP GET request to the service end point appended with a cultural object ID and a “.json” extension.

### Request Examples

* GET https://data.nga.gov/art/tms/objects/1138.json

### Constraints and Assumptions

* The cultural object id will be URL escaped if necessary, e.g. if it contains characters reserved for use in a URL

### Processing the Request

* Read the ID, unescape if necessary, look up the object, return the cultural object record or an error

### Response

* A single JSON formatted unabridged Cultural Object Record with a default namespace of “cultObj”
* Note that the predicate is specified from the perspective of the entity in the search results, aka the “item”, and the object of the predicate is defined in the reference structure

### Response Example

{

"record": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods",

"artistNames": "Bellini, Giovanni (artist); Titian (artist)",

"accessionNum": "1942.9.1",

"bibliography": ["Hartshorne, Rev.C.H.A Guide to Alnwick Castle.London, 1865: 62.", "<i> Paintings in the Collection of Joseph Widener at Lynnewood Hall. < /i> Intro. by Wilhelm R. Valentiner. Elkins Park, Pennsylvania, 1923: unpaginated, repro., as by Giovanni Bellini."],

"displayDate": "1514 / 1529",

"creditLine": "Widener Collection",

"description": "The Feast of the Gods, Giovanni Bellini and Titian, 1514 / 1529",

"location": ["1942.9.1 – West Building Main Floor Gallery 12"],

"homeLocation": ["1942.9.1 – West Building Main Floor Gallery 12"],

"inscription": "lower right on wooden tub: joannes bellinus venetus / p MDXIIII",

"medium": "oil on canvas",

"departmentAbbr": "DCRS",

"dimensions": "overall: 170.2 x 188 cm(67 x 74 in .) \n framed: 203.8 x 218.4 x 7.6 cm(80 1 / 4 x 86 x 3 in .)",

"provenance": "Probably commissioned by Alfonso I d 'Este, Duke of Ferrara [d. 1534);[1] by inheritance to his son, Ercole II d'Este, Duke of Ferrara[d .1559];by inheritance to his son, Alfonso II d 'Este, Duke of Ferrara [d. 1597]; by inheritance to his cousin, Cesare d' Este...",

"references": [

{ "predicate": "hasPrimaryDepiction",

"object": {

"namespace": "image",

"source": "intranet",

"classification": "photography",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef"

"fingerprint": "d63a8ffddac498cb861a53e11989cef"

}

},

{ "predicate": "hasChild",

"object": {

"namespace": "cultObj",

"source": "tms",

"id": "1234",

"title": "",

"classification": "frame"

}

}

]

}

}

### Error Handling

* implementation shall return a 404 not found error if the ID does not correspond to a cultural object in the partner CMS/CMSes
* if no source is specified, this service should redirect to the search for cultural objects records service to execute a search for the id provided or it should return a 400 Bad Request error with an error structure as defined in the error logging service
* if a malformed request is received (doesn’t contain a set of parameters that can be understood by the service or is missing mandatory parameters such as the object id an error 400 Bad Request shall be returned
* in case of any other internal error, a 500 error response shall be returned
* whenever an error condition occurs, the body of the response may be empty or, ideally, it should consist of a JSON error message structure as defined in the error logging service, e.g. severity, summary, details, origin – providing this structure will make it easier for Sirma and partners to diagnose problems with the interface
* In general, if a request cannot be mapped to any of the supported services, a 404 Not found error should be sent. if ConservationSpace receives an incomplete response from the service, an error message shall be posted to the logger service with details about the error. In order to avoid error loops, ConservationSpace shall not report an error encountered with the error logger service itself and it shall detect redirect loops and terminate a request if one is discovered.

## Service #2: Search for Cultural Object Records

### Response

Returns a cultural object record as defined in *Result Set: Records* with records for items corresponding to an ***abridged cultural object*** record which is defined as the mandatory set of properties including references to other entities and all cultural object properties used in the partner’s cultural object search header template.

### Request

HTTP GET or POST request to the service end point. ConservationSpace will use the GET method except in cases when the query string would cause the URL length to exceed the maximum length as defined in the HTTP specification (more than 2000 symbols of URL encoded data). ConservationSpace will provide the search queries and pagination configuration in the query string or in the posted form data.

### Request Examples

* GET https://data.nga.gov/art/objects.json?number=1987.1.&title=blue&skip=0&limit=10&orderasc=number&orderdesc=title
* GET https://data.nga.gov/art/objects.json?lastModified=2016-01-01T00:00:00.000Z&lastModified=2016-01-10T00:00:00.000Z&skip=20&limit=10
* POST /art/objects.json?skip=20&limit=10 HTTP/1.1

Host: data.nga.gov

Content-Type: application/x-www-form-urlencoded

Content-Length: 76

lastModified=2016-01-01T00:00:00.000Z&lastModified=2016-01-10T00:00:00.000Z

### Constraints and Assumptions

* search operators are fixed for each of the five search criteria (see table below)
* if “source” is supplied in the URL, the search shall be constrained to only that source – if source is not supplied, all sources should be searched
* the search implementation shall combine all supplied constraints with the Boolean "AND" operator
* with the exception of the lastModified attribute, multiple values for search criteria will not be sent to this interface and need not be supported within an implementation – if multiple values are supplied for attributes other than lastModified timestamp, only the first value needs to be used. *Partners may optionally implement support for multiple valued criteria by creating a sub-query for that field which ORs the multiple values together, however, ConservationSpace will not supply any multi-valued search criteria except for the “lastModified” timestamp for which a maximum of 2 values will be sent in ConservationSpace R2*
* If a single value is supplied for lastModified and that value is empty, it should be ignored. If a single non-empty value is supplied, that value should be assumed as the earliest date with an unbounded upper limit. When two or more values are supplied, only the first two values should be used and those values represent a date range. If the second value is earlier than the first, the values should be swapped by the API implementation in order to construct a valid date range for the search. If the first value is empty and the second not-empty, an unbounded minimum shall be used for the search.
* skip has a default value of 0; limit has a default value of 50 and a maximum value of 1000; order shall default to alphabetical by cultural object title
* order can be used to control the order of the result set – order should consist of one or more comma delimited search fields with descending order specified by prefixing the field name with a minus character “-“ (see examples in table below). If no sort order is specified, a partner-specific default ordering should be used although it is suggested that Object Number / Accession Number be used if such numbers are available. For usability concerns, it is recommended that the default ordering only use fields present in the search header. *Since ConservationSpace will only supply a primary sort criteria, support for secondary, tertiary, etc. sorts are optional.* The NGA intends to support them and will append the default sort criteria (accession number ascending) to the end of each requested sort order in order to differentiate all items in the result set.
* all search criteria values will be properly escaped by ConservationSpace so they can appear in query strings -they will need to be unescaped before using – most programming frameworks will unescape query strings and form parameters automatically

|  |  |  |  |
| --- | --- | --- | --- |
| **Request Property**  (default namespace is "cultObj") | **Description** | **Matching Operator to Use** | **Example of Value(s)** |
| id | The unique cultural object identifier for the tenant’s CMS, e.g. TMS ID, CONA URL, etc. | equals | 12345 |
| lastModified | The last modified date of the art object – if this is not known or a tracking mechanism cannot be built, it will be impractical or inefficient for your institution to automatically update cultural object metadata in ConservationSpace and such updates might have to be initiated by a ConservationSpace user. ConservationSpace will send ISO 8601 compliant dates as shown in the example, but there is no harm in supporting additional date formats. The NGA computes a “last detected modification” timestamp as part of a data extraction process from TMS and we are happy to discuss the implementation details or to share the PLSQL and views used in the TMS extraction process. | between | 2016-01-01T00:00:00.000Z  2016-01-10T00:00:00.000Z |
| number | The accession number or similar institutional tracking number. | contains | 1987.1. |
| title | Searches title(s) of the art objects | contains | The evangelist Matthew and the angel |
| artistNames | Searches names of artists associated with the work and possibly alternate names, although that will be a tenant implementation decision | contains | Rembrandt |
| order | Parameter specifying the sort order to use in the response with ascending order being the default and descending order being specified by prefixing the desired field with a minus character “-“. All searchable fields MUST support sorting although the implementation may choose to implement the sort using another field or complex logic. For example, if artistNames is selected as the primary sort order and the artistNames field consists of (First Name / Last Name), the implementation may opt to sort by the last name of the first artist instead of the first name. *Partners may optionally implement support for secondary, tertiary, etc. sorts and if they do, the fields must be separated with a comma. ConservationSpace R2 will only request primary sorting. Non-primary sorting may also be added programmatically after the primary sort if so desired.* | N/A | order=artistNames,-lastModified,cultObj:id  for multiple ordering criteria  or  order=cultObj:number  for a single ordering criteria |
| references | A Boolean parameter that defaults to true and is used to suppress related entities (“references”) from appearing in search result items. This is provided to give ConservationSpace the opportunity to improve performance by excluding references when none are needed in response to a search. | N/A | 0, 1, false, true |
| thumbnails | A Boolean parameter that defaults to true and is used to suppress thumbnails in the response when they are not needed. Computation of base64 thumbnails can be an expensive operation so ConservationSpace shall suppress them when they are not necessary, e.g. when determining the set of imported art objects that have been modified since last updated. | N/A | 0, 1, false, true |

### Processing the Request

* the search query will always consist of at least one constraint, but may contain up to five
* With the exception of date ranges, search properties with blank values should be ignored
* all valid constraints should be combined with an implicit Boolean “AND”
* wildcard characters and regular expressions are not intended to be supported and all characters may be treated as literal values – institutions wishing to support wildcard characters may do so although no wildcard characters standard has been developed for ConservationSpace at this time
* all searches should be case insensitive even if provided values contain a mix of upper and lower case characters
* sorting of the result set should occur prior to applying pagination
* optional: if possible, when a diacritical variation of a character is provided, e.g. when e is provided in a search query and there exist diacritical forms such as é in the data being searched, the search should include results for both the diacritical and non-diacritical forms; however, when a diacritical form is provided, only results matching the diacritical form should be returned
* if no search criteria are supplied, the implementation should return an empty result set rather than returning all records

### Response

JSON formatted *Result Set: Records* where each item is a result set with properties corresponding to an ***abridged cultural object record.***.

### Response Example

{ "items": [{

"url" : "https://data.nga.gov/art/tms/objects/1138.json",

"record": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods",

"artistNames": "Bellini, Giovanni (artist); Titian (artist)",

"accessionNum": "1942.9.1",

"displayDate": "1514 / 1529",

"location": ["1942.9.1 – West Building Main Floor Gallery 12"],

"references": [

{ "predicate": "hasPrimaryDepiction",

"object": {

"namespace": "image",

"source": "intranet",

"classification": "photography",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef"

"fingerprint": "bdacd63a8ffd498cb861a53e11989cef"

}

},

{ "predicate": "hasChild",

"object": {

"namespace": "cultObj",

"source": "tms",

"id": "1234",

"title": "Frame of The Feast of the Gods - F232.2134",

"classification": "frame",

"artistNames": "Franz Rafthouser (painter)",

"accessionNum": "1942.9.2",

"displayDate": "1729",

}

}

]

},

"thumbnail" : "https://vm-imgrepo-tdp.nga.gov/public/objects/1/1/3/8/1138-primary-0-90x90.jpg"

}],

"paging": {

"limit": 1,

"skip": 50,

"total": 176

}}

### Response

* The API shall return a JSON formatted *Result Set of Records* where every item is a result set with records corresponding to an ***abridged cultural object***.
* The thumbnail shall be either a URL accessible via web browser to each of the partner’s ConservationSpace users or it should consist of the base64 encoded thumbnail’s binary contents. ConservationSpace will decode base64 images and serve them through the ConservationSpace web server.

### Error Handling

* if the request contains unexpected input or formats, a server 400 Bad Request HTTP error code should be returned to the caller
* if no search results exist, the result set should be empty (total:0) as opposed to issuing an error code such as a 404 file not found error
* if ConservationSpace receives an incomplete response, an error will be posted to Service #8 – the logger service with details about the error

## Service 3: Retrieve Media Content by ID

returns the binary content of a media asset based on its source, media type, and ID

### Request

HTTP GET request to the service end point appended with an media ID

### Request Examples

* GET <https://data.nga.gov/media/intranet/images/d63a8ffd-bdac-498c-b861-a53e11989cef> (“images”is the media type)

### Constraints and Assumptions

* Only image transfers will be supported in ConservationSpace R2 and therefore the only media-type issued by ConservationSpace will be “images”. However, in the future, this service could be used to fetch additional media content and therefore multiple types of media would need to be supported

### Processing the Request

* The supplied media ID is provided by the media search API and should therefore be able to be found in the partner’s DAMs for the given source system of record and the specified media type
* If the network configuration with ConservationSpace enables the media to be downloaded directly by ConservationSpace, the service may issue a redirect to the corresponding URL and ConservationSpace will download the media asset directly from the source rather than functioning as a proxy
* If ConservationSpace does not have direct access to the media asset over http, then this service should read the media contents from the source and stream the bits directly back to ConservationSpace using the appropriate content-type HTTP header based on the mime-type of the media asset’s type and format.

### Response

* Binary media content with the content-type set to the proper mime-type for the media asset, e.g. *image/ tiff* as defined at <http://www.iana.org/assignments/media-types/media-types.xhtml>. The proper content-type MUST be set in the response.

### Response Example

Content-Type: image/jpeg

Content-Length: 400

9c c2 89 e5 a1 dd 0a 3e 73 38 c5 ec 94 a6 c3 6b

ae 28 3e 58 b0 b3 f0 bc 5c b6 ed 8e 11 e9 34 20

34 24 81 a5 59 5d 8d 4d 14 f0 df 1d 5d 04 eb 1f

c2 53 80 1d 0c be f4 90 a4 38 10 d6 9f 86 1f 64

2e 6f 96 9c 6e 91 10 79 7c 5b 4a 3d 09 dc 33 f7

88 7c 06 76 87 44 b4 6a 1a c6 c6 0c 29 e1 51 ef

87 1f 8b c9 0b 43 ec 95 1a b1 fc 3b 09 8f 72 40

be 4f 9d ba 3d db 09 d8 ba db 23 65 b8 35 5a 8c

65 f7 de f9 b1 67 68 46 60 3c c3 62 69 ca e0 82

18 f6 fa e3 b0 77 7b 47 26 ef 46 19 8d 90 d2 7a

f1 ee 2c 62 26 25 3e f5 d5 69 8f 7d 45 81 58 bf

56 91 1b f8 1d 03 cb 95 cc 06 a9 c2 e5 5f 0e 0f

Note: hexadecimal representation is for illustrative purposes only to convey the notion of binary data transfer

### Error Handling

* If no source is specified in the URL, this service should return a 400 bad request error
* implementation will return a 404 not found error if the ID does not correspond to a media itemfor the given source and media type
* if partners receive an incomplete or improperly formatted request a 400 Bad Request shall be returned to ConservationSpace
* if ConservationSpace receives an incomplete or unexpected response, an error will be posted to the error logger service with details about the error

## Service 4: Retrieve Media Record by ID

Retrieves media records based on the supplied source, ID, and media type and returns an unabridged media record.

### Request

HTTP GET request to the service end point appended with the media type and a media ID followed by a “.json” extension. The JSON format for the media type “images” is defined under *Record: Image* in this document. Other media types may be defined in the future, but ConservationSpace R2 will only utilize the images media type.

### Request Examples

GET https://data.nga.gov/media/intranet/images/d63a8ffd-bdac-498c-b861-a53e11989cef.json

### Constraints and Assumptions

* The namespace “image” and media type “images” shall be the only media namespace and type that needs to be supported in ConservationSpace R2, although future releases might introduce additional media types, each with a corresponding namespace, e.g. “audio”, and record definitions.

### Processing the Request

* The supplied media ID is provided by one of the search APIs and should therefore be able to be found in the partner’s DAMs for the given source and media type.

### Response

* implementation will provide a single unabridged JSON media record.

### Response Example

{

"record": {

"namespace": "image",

"source": "intranet",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef",

"classification": "image",

"mimetype": "image/tiff",

"title": "60-primary-0-nativeres.ptif",

"lastModified": "20160421T182434-04:00",

"fingerprint": "3537aac2467dce2e146e8195364e2c5f34bf1aead29b1fbbe91c",

"pixelsPerCM":489.34325,

"lightType": "Normal Light",

"orientation": "Front",

"references": [{

"predicate": "primarilyDepicts",

"object": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods",

"artistNames": "Bellini, Giovanni (artist); Titian (artist)",

"accessionNum": "1942.9.1",

"displayDate": "1514 / 1529",

}

}]

}

}

Error Handling

* implementation will return a 404 not found error if the ID does not correspond to a media asset in the DAMs for the given source and media type
* if a source is not specified in the URL, the service may redirect to the search for media records service or return a 400 “Bad Request” error with an error message formatted as defined in the error message service
* if a malformed request is received, an error 400 Bad Request is returned
* if ConservationSpace receives an incomplete response, an error will be posted to the logger service with details about the error

## Service 5: Search for Media Records

Returns a *Result Set: Records* response where each item consists of an abridged media record , i.e. all mandatory properties for the media type including references to other entities and all fields used in the partner’s media search header template. Only the media type “images” needs to be supported in ConservationSpace R2.

### Request

HTTP GET request to the service end point. Caller will provide the search queries and pagination configuration in the query string

### Request Examples

* GET https://data.nga.gov/media/images.json?cultObj:id=1138
* GET https://data.nga.gov/media/intranet/images.json?cultObj:number=1987.1.&skip=0&limit=10&orderasc=cultObj:number&orderdesc=cultObj:title
* GET <https://data.nga.gov/media/images.json?id=d63a8ffd-bdac-498c-b861-a53e11989cef&skip=20&limit=10>
* POST /media/images.json?skip=20&limit=10 HTTP/1.1

Host: data.nga.gov

Content-Type: application/x-www-form-urlencoded

Content-Length: 128

lastModified=2016-01-01T00:00:00.000Z&lastModified=2016-01-10T00:00:00.000Z&id=UADFA3908234234&id=ASDFIASDFOKJ23423&id=12309823423843ADFASDF…

(presence of many IDs mandates that POST be used rather than GET)

### Constraints and Assumptions

* search operators are fixed for each search criteria (see table below)
* if source is specified in the URL, the search should be limited to that source, otherwise all sources should be searched
* the search implementation shall combine all supplied constraints with "AND"
* if unspecified, results may be ordered in a partner determined way but preferably using only fields that appear in the media search header template
* like cultural object searches, ordering of media search results should be supported for each of the search fields for the given media type with optional implementation for non-primary sort orders and default ordering appended to the requested order to differentiate otherwise equivalently sorted items
* all search criteria values will be properly escaped so they can appear in query strings and will need to be unescaped before using
* in ConservationSpace R2, the default namespace for this service shall be “image” since that’s the only supported media type. Cultural object properties supported in the media search shall always be prefaced with “cultObj:” or they will be misinterpreted as image properties or just ignored rather than being treated as cultural object properties.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Request Property** | **Description** | **Matching Operator to Use** | | | **Example of Values** | | |
| id | The media (for now image) identifier | equals | | | 12345 | | |
| lastModified | The last known modification date of the media asset | between | | | 2016-01-01T00:00:00.000Z  2016-01-10T00:00:00.000Z | | |
| cultObj:id | The unique cultural object identifier for the partner’s CMS, e.g. TMS ID, CONA URL, etc. | equals | | | 12345 | | |
| cultObj:number | The accession number or similar institutional tracking number. | contains | | | 1987.1. | | |
| cultObj:title | Title(s) of the art objects | contains | | | The evangelist Matthew and the angel | | |
| cultObj:artistNames | Names of artists associated with the work and possibly alternate names, although that will be a tenant implementation decision | contains | | | Rembrandt | | |
| order | Parameter specifying the sort order to use in the response with ascending order being the default and descending order being specified by prefixing the desired field with a minus character “-“. All searchable fields MUST support sorting although the implementation may choose to implement the sort using another field or complex logic. For example, if cultObj:artistNames is selected as the primary sort order and the artistNames field consists of (First Name / Last Name), the implementation may opt to sort by the last name of the first artist instead of the first name. | N/A | | | | order=cultObj:artistNames,-image:lastModified,id | |
| references | A Boolean parameter that defaults to true and is used to suppress related entities (“references”) from appearing in search result items. This is provided to give ConservationSpace the opportunity to improve performance by excluding references when none are needed in response to a search. | | N/A | 0, 1, false, true | | |
| thumbnails | A Boolean parameter that defaults to true and is used to suppress thumbnails in the response when they are not needed. Computation of base64 thumbnails can be an expensive operation so ConservationSpace shall suppress them when they are not necessary | | N/A | 0, 1, false, true | | |
|  |  |  | | |  | | |

### Processing the Request

* The ConservationSpace UI for the media type “images” includes separate search boxes for image ID and object ID. ConservationSpace will send “cultObj:id” when an ID has been entered into the cultural object ID search box.
* when object related search criteria are sent, the implementation may opt to execute a CMS search to locate the relevant object IDs and then return all media records for the specified media type associated with those cultural object IDs; however, if robust metadata is already available within the DAMs, it may be simpler to execute a search of cultural object metadata in the DAM; this will be a partner implementation decision
* if no search criteria are specified, the implementation should return an empty result set rather than returning all records
* unless the search property is an object id, media id, or a date range, only the first occurrence of the value shall be used
* the handling of multiple values provided for a search field mirrors that of the cultural object search
* search properties with blank values should be ignored except for date ranges which should be treated identically as specified in the cultural object record search
* if the lastModified date of a media assets not available, a reasonable time stamp alternative such as createdOn should be substituted when executing a lastModified search; in such cases, the label associated with the lastModified field should be adjusted in the partner’s configuration in ConservationSpace to a more accurate description all valid search criteria should be combined with an implicit Boolean “AND”
* include relationships to other entities only if the references parameter is unset or true
* include thumbnails only if the thumbnails parameter is unset or true
* wildcard characters and regular expressions are not intended to be supported and all characters maybe treated as literal values; partners may implement support for wildcard characters if they like, although there have been no discussions about ConservationSpace wildcard character standards
* all searches should be case insensitive
* sorting of the result set should be applied prior to pagination

### Response

JSON formatted *Result Set: Records* where each item is an abridged media record

### Response Example

{"items": [

{ "url" : "https://data.nga.gov/media/intranet/images/d63a8ffd-bdac-498c-b861-a53e11989cef.json",

"record": {

"namespace": "image",

"source": "intranet",

"id": "d63a8ffd-bdac-498c-b861-a53e11989cef",

"classification": "image",

"mimetype": "image/tiff",

"title": "60-primary-0-nativeres.ptif",

"lastModified": "20160421T182434-04:00",

"fingerprint": "35467dce2e146e8195364ade2c5f34bf1aead29b1fbbe91c",

"pixelsPerCM":489.34325, "lightType": "Normal Light",

"orientation": "Front",

"references": [{

"predicate": "primarilyDepicts",

"object": {

"namespace": "cultObj",

"source": "tms",

"id": 1138,

"classification": "painting",

"title": "The Feast of the Gods",

"artistNames": "Bellini, Giovanni (artist); Titian (artist)",

"accessionNum": "1942.9.1",

"displayDate": "1514 / 1529",

}

}]

}, "thumbnail" :

"iVBORw0KGgoAAAANSUhEUgAAAHIAAAFHCAYAAAEKGihBAAAAAXNSR0IArs4c6QAAAARnQU1BAACx

jwv8YQUAAAAJcEhZcwAADsQAAA7EAZUrDhsAADbDSURBVHhe7V0JXFTVF56yzCz3fUf2TU0tc03N

TC0XhGFVAVFRYGaYYVFEE1MzFXfLNHdzwX1NTXAXgWGZART3XDK39K/mlll9/3PezBASKiDgML7v

9/t4b4Z5c+e87517z73vvnMlRQqZXKmWyVXz5XLVI/1bBcIGYn2iDbE/v1Eg7D/x2wf63YIh/ZcH

SDh9B/qXBUPmrw9f7ODEzHMvqeQXPPj2yyr5zN3CHXz6KrAnXiscLMvyhremp+GLphAHEH2EV3nB

Nfk9SA/bZZfsPN8GCoVSFRQU1iYoKKiuXBESK1OoAvX/fhJ8cPgRt8L9bD7YNd7+JR08L+HrQh6s

phMW/+8JKxBe6Ge70cEu8TaFP/iFSi60zS9c8gsdLI/v9TJKpovkhX521JEhL+mEvdDBTvEWL6nk

vvFWpeiELeM/7iktMHv3VwUv2T3Z7tj4zFHwSm2Hjj80LNgX8EHu9LM9k5vgo+UNUcFPiEvyh95x

NnBXO6L3LmvUUL5R8J9eK+RNVPSRoJKfpHBnvMCQK5Tz5MHKqbJg1Wz9W/mCmuin25VEEl/T7eYT

x365/6F+t2B4/Pc/mLHpTMFPDh/4dczJwh0YezCthEss9IEzNxwvwRL/Bp440PWoa1naVNa9egZ+

uXEJ0et/hXtCR7MgrQ+kaR2m0duf6f4rWULkoGyQ8ConojO/QtNp1YQSnebYVus13XJaWHgEZArl

JIUiLEomU+Tdh1imXYwW02sX3EYusYm+xAKh0AeuS1uDdftiClFiBpdYtQR/6pTMCXBd2KXgB07N

nJitY4HABxbKRt2BhS1xeiEPbDO7cQmfnKaF/anr9q4u4Z9a6LP64awCNuuMQpfIF7nd1IqFK9Fh="

}

],

"paging": {

"limit": 1,

"skip": 50,

"total": 176

}}

### Error Handling

* if the request contains unexpected input or formats, an HTTP error code 400 Bad Request error should be returned to ConservationSpace
* if no search results exist, the result set should merely be empty – a 404 should NOT be returned for this condition
* if ConservationSpace receives an incomplete response, an error will be posted to the logger service with details about the error

## Service 6: Error logger

Creates log entry with detailed error messages and severity for operations and debugging purposes

### Request

HTTP/S GET or POST request with four parameters as outlined in the response section below

### Request Examples

* POST https://data.nga.gov/system/logger.json

Host: data.nga.gov

Content-Type: application/x-www-form-urlencoded

Authorization: QWxhZGRpbjpPcGVuU2VzYW1l

Content-Length: 116

severity=error&origin=someurl&summary=maximum%20value%20for%20pageSize%3D25&details=The%20limit%20for%20page%20size%20is%2025%20since%20...

* GET https:// data.nga.gov/system/logger.json? severity=error&origin=someurl&summary=maximum%20value%20for%20pageSize%3D25&details=The%20limit%20for%20page%20size%20is%2025%20since%20..

### Constraints and Assumptions

* ConservationSpace will use a POST request to this service by default
* Severity should be one of *fatal, error, warn, info, debug, or trace*

|  |  |  |
| --- | --- | --- |
| **Request Property** | **Description** | **Example of Values** |
| severity | fatal | error | warn | info | debug | trace | “error” |
| origin | String representing the origin of this message, e.g. the request that resulted in the message needing to be logged, a transaction number that can be traced to a source, etc. | “http://data.nga.gov/art/objects/1138.json?title=B89” |
| summary | Short version of the message to be logged | “Maximum value for page size is 25” |
| details | Long version of the message to be logged | “The limit for page size is 25 since …” |

### Processing the Request

* The implementation may optionally authorize the authorization header which will only be sent if the HTTPS protocol is in use
* All parameters will be escaped as per the HTTP protocol
* As with other services, the security token defined in the tenant configuration will be sent with every logger request over HTTPS

### Response

* On successful receipt of log message and recordation of the message, implementation shall respond to ConservationSpace an HTTP status 200 OK and the body of the response should contain a simple JSON structure that includes the severity, origin, summary, and details provided via the query string (if a GET request was used) or the posted form data if a POST request was used. This will allow ConservationSpace to confirm that its message was received successfully by the target system.

### Response Example

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 116

{

"severity" : "info",

"origin" : "sampleorigin",

"summary" : "mysummary",

"details" : "mydetails"

}

### Error Handling

* An HTTP 401 if an authorization problem occurs or 400 Bad Request if message does not conform to the required format or contain the expected parameters.
* If the severity is not recognized, an HTTP 400 Bad Request error should be issued

# ConservationSpace Provided Service Endpoints

The ConservationSpace user interface utilizes a series of RESTful services to perform all user operations. These services are available to programmers who wish to perform automated operations in ConservationSpace.

The current version of ConservationSpace RESTful API documentation can be found in Confluence at the link below. This documentation will evolve as Sirma continues refining the new user interface.

https://www.wiztechno.com/wiki/display/CSR2/CS+Services

## Security of the ConservationSpace Provided Services

Partner systems can authenticate against the services by either:

* Using header ‘ssoToken’ with value a valid SAML 2 token requested from the ConservationSpace IDP – the acquisition of SAML tokens will be outlined in the wiki when the services are available
* Using header ‘username’ and ‘password’ with the desired login credentials. We strongly recommended using a dedicated user account for such operations.

# Integration Components within the ConservationSpace System

## EAI Design

The design of the ConservationSpace platform’s side of the integration with partner’s API implementation is based on an external system id and tenant aware configuration. The design is driven by the idea to reuse existing services and domain model in the system.

Each subsystem (CMS, DAM) has tenant specific extensions as services, configurations and model providers. All EAI extensions implement Named interface so to be runtime distinguishable. The returned values of its method getName are either CMS or DAM.

The base extension to configure each external subsystem is - EAIConfigurationProvider. It must provide the communication endpoints, model configurations and other tenant+system specific settings using other extensions.

EAIConfigurationProvider has a method getSystemClientId that returns the current subsystem+tenant specific id so to be runtime distinguishable.

## Communication

To provide flexibility during request preparation, communication and response parsing are provided with a set of extensions for each subsystem. Then the specific logic and models are separated in the required module only.

List of extensions to implement for each subsystem:

* EAICommunicationServiceAdapter - provides communication capabilities using a specific request model and specific response model. The protocols, models and how services are invoked could be customized
* EAIRequestProviderAdapter - provides customized request to be consumed by the CommunicationServiceAdapter.
* EAIResponseReaderAdapter - provides customized response parsing and preparation of data to be sent to the invoking code in CS. Loads the existing items from response, or creates temporary instances

There are three general services: Search, Retrieve (import), Log that are accessible as ids in the system. Each service id is based on subclass of EAIServiceIdentifier.

## Dynamic model

### Mappings

Model configuration is needed for several tasks: convert model CS <-> external subsystem, validation of response model during conversion, provide information about the CS model based on provided mapping, provide information about the relation model.

Model is currently based on external xlsx models based on wiztechno pages with model (See wiztechno page for more details). To configure a system+tenant, several xlsx files should be placed on the server with following structure: subsystem/tenantid - i.e. cms/nga.gov, cms/smk.com, dam/nga.gov. The minimum required files are types.xlsx containing all the supported types for this particular system and tenant, common.xlsx containing the common properties for all object types and for each type from types.xlsx a new $(id).xlsx that holds the properties configuration for each type and relations.xlsx for the relation information.

The parsed model is wrapped in sealed bean ModelConfiguration. It contains a list of EntityTypes, each holding a list of EntityProperties and EntityRelations. For each subsystem+tenant there is a wrapper class (ModelConfiguration) holding all that information.

EntityType contains at least the external name (i.e. Painting) and the uri in CS (NGACO7002).

EntityRelation contains the mapping of single relation to a type based on the entity title or the keywords 'all', 'Entity' which means that the relation is added to all processed types.

For properties contained in commons.xlsx and the specific type (i.e. NGACO7002) the columns specified by the configuration property 'eai.cms.config.model.xlsx.mapping' and 'eai.dam.config.model.xlsx.mapping' are processed and extracted. If row is with italic font or it is importable (column 2 is Y) the row is added as EntityProperty to the EntityType. Properties from the specific type, might override properties from commons.xlsx.

The folder with models for CMS is specified by the 'eai.cms.config.model.xlsx.path' configuration and for DAM is 'eai.dam.config.model.xlsx.path'.

### Read the Data Mapping Model for a Tenant

It is possible to use the ConservationSpace services to access the current schema used for mapping between interface parameters and cultural object / image ontologies. The response is a JSON array of entity types such as (Drawing, Painting, etc.) containing the properties and configurations (mandatory/optional, property data types, etc.), their associations and configurations. The resulting JSON structure will include classification-specific models that can be used for determining when to include certain types of properties in a response.

### Request (GET)

https://<cspace host>/emf/service/integration/cms/model

https://<cspace host>/emf/service/integration/dam/model

### Request Example

GET /integration/CMS/model

### Response Example

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Lenth: 8823

[{"name":"Drawing","relations":[{"name":"belongs with", "mandatory":false, "multivalue":true},...

### Programmatic Update of Tenant Model

This is a push model, instead of pull and can be used for batch import of images and cultural objects. This service is intended to read the same format as *Service #2: Search for Cultural Object Records* and *Service #5: Search for Image Records* containing a full model for a cultural object or image. This document will be updated with more detail as this service becomes available.

### Request (POST)

https://<cspace host>/integration/<CMS|DAM>/import

### Request example

See full details in #3, #7

Data is JSON object with one “items” field mapped to array of objects. Paging in not supported in this request

POST /integration/DAM/import

POST /integration/DAM/import

Content-Type: application/json

{"items": [ { "url" : "https://data.nga.gov/media/images/intranet/1138-primary-0-native.ptif.json", …

### Response

Response contains either status code 200 or error message wrapped with specific status code

### Error Handling

* An HTTP 401 if an authorization problem occurs
* An HTTP 500 if fail to process request

### Model Converter

Converter is used for two-way conversion of data - CS<--->external system.

CS -> External system

Conversion of CS property to external property is based on 4 params:

- key - the property name;

- value - the property value;

- defintionId - the defintion model to use;

- usage - the entity property mapping ( search criteria/data mapping);

If the codelist contains the property, then the new value is extracted from the codelist. Values from extra2 and extra3 are used for CMS and DAM.

External system -> CS

Conversion of external property to CS property is based on 2 parameters:

- externalProperties - the properties for conversion. If their value is not serialized a conversion to string is triggered; Supported values are strings, numbers, dates. Conversions is based on the mapped internal CS model.

- instance - the instance that will consume the properties; It is used to find the correct mapping. The instance type is taken from the classification value from response.

Conversion of external property to CS might be handled in different ways:

- if external property exists in codelist, then the CS values are extracted from extra2, extra3 values; If value is missing it is either ignored or other action listed in the specification is taken.

- type conversions is based on the internal CS model (the exact definition and property that matches the external property). Property multiplicity is also matched from definition model.

During model conversion, several validation steps are taken: check mandatory values, check CL value validity, check multiplicity. All data is compared to the current definitions model for that particular type (Painting, Image, etc.)

## Search

Search in external system is based on SearchEngine extension that checks if there is a context set in the request param. If the context is set and matches one of the registered external systems (CMS, DAM) then the request adapter is used to prepare the query.

### Prepare request

The request (query) is based on three params:

* systemId - the external system id (CMS, DAM);
* serviceId - the id of the service (search);
* searchArguments - the arguments used during request generation;

### Execution/Invocation

After preparing/generating the request the CommunicationService is used to execute it and to generate a response. During the invocation, response is processed (paginating and result ordering) and then it is ready to be parsed.

### Parsing response

The ResponseReader is used to prepare the internal system model from the generated response. Each type of response generates potentially different type of the parsed response. The parsing is system specific. If instance is unknown (external id + source id tuple is missing the semantic db) a temporary object is created using the operation "prepareImport". This operation leads to specific status of the created instance ( FOR\_IMP ).

## Import

Import is done using the retrieve service in same three step manner (prepare request, execute, parse response) plus processing the import response.

The import service URI is currently POST@ service/integration/import. The payload is collection of Instances. Each instance should contain the integration properties: integratedSystemId, emf:externalID, chc:sourceSystemId.

During the import the instance that is executed import on is processed and needed metadata is extracted (integratedSystemId, emf:externalID, chc:sourceSystemId). Then the retrieve service is invoked with the extracted parameters.

### Processing the import response:

The import request is processed by IntegrateExternalObjectsService service, which relies on adapter for each subsystem to process the storage (instance of IntegrateObjectsServiceAdapter). During import object is created using the operation "import" and it is persisted in db.

Relations are created on import when referenced item exists (reference to) in database, otherwise reference is skipped. Other relations are created based on last specifications (as primaryImage, etc.).

During import of images, image content is scheduled for asynch download and it is processed in background after some time. This allows to import several images simultaneously with varying size. On the other hand the image document/object is imported and persisted.

## Update

### Automatic update

Automatic update of objects like every other automatic task in CS is handled by timer definition/script.

The execution time of the timer depends on the cron expression set in the control params of the definition.

The cron expression can be generated in http://www.cronmaker.com/

The timer executes script which performs the following steps:

* search for all updated objects in external system for last 24h by their lastModifiedOn property;
* re-import/update the object in CS;

In generic definitions folder there is a file named ngaTimedActions.xml where auto update script is defined. If the user wants to create another timer, he has to add another "field" tag with unique name and different script. The timer schedule might be configured with control tag which has id="schedulerConfiguration".

# Examples for configuring auto update schedule. The cron expression can be generated in http://www.cronmaker.com/.

# 1) Setting automatic update schedule to fire every day at 00:00h

<control id="schedulerConfiguration">

    <control-param id="config" name="cronExpression">0 0 0 1/1 \* ? \*</control-param>

</control>

# 2) Setting automatic update schedule to fire every Sunday at 01:00h a.m.

<control id="schedulerConfiguration">

    <control-param id="config" name="cronExpression">0 0 1 ? \* SUN \*</control-param>

</control>

### Manual update

The manual update of objects will be triggered manually by user. The user will be able to update the object only if the object is already imported. The update action will invoke the import service to trigger re-import/update for the current item.

## Persistence

Models is based on the internal domain model in the system. Using this approach each item from response is represented as instance in the system. Each instance that is not imported is persisted in temporary storage in runtime cache. This way all logic regarding permissions, definitions, visualization and so on is directly reused. The already persisted instances are loaded with a query based on externalId + sourceSystemId and are merged with the queried instances' metadata.

During import temporary instances are retrieved from the external system and are stored in db. The status of instances is FOR\_IMP.

## Error handling

* EAIReportableException is reported the remote system on any failure with the wrapped cause. Such exceptions may occur on: request failure, transformation of response failure, other relevant information.
* EAIModelException indicates an error during model parsing/generation.
* EAIRuntimeException indicates an error in any phase of integration process.

During search and update/import errors are gathered for each instance if the error is not critical. Critical error in data is considered to be: missing externalId, sourceSystemId, namespace. The consolidated error is returned to the user.

## Example Configurations

The following list of current configurations is for example purposes. There are tenant specific configurations in the database for each tenant.

#for cms

eai.cms.communication.uri=http://localhost:12000/partner

# the service uri with timout setting. the default value is enough in most cases

eai.cms.communication.services = {"search":{"GET":{"uri":"/art/objects.json", "timeout":72000}},"retrieve":{"GET":{"uri":"/art/{0}/objects/{1}.json", "timeout":-1}},"logging":{"POST":{"uri":"service/log", "timeout":12000}}}

# the specific token that is provided from partner

eai.cms.communication.security.token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

# set to false to honor only trusted certificates

eai.cms.communication.security.trust.all.certificates=true

# path for models as specified in design

eai.cms.config.model.xlsx.path=/opt/wildfly-9/config/eia/cms

# mapping of xlsx data to its value. In most cases default value is enough.

eai.cms.config.model.xlsx.mapping={"TITLE":0,"URI":12,"PPROPERTY\_ID":13,"MAPPING\_DATA\_CONVERT":14,"MAPPING\_CRITERIA\_CONVERT":15,"MANDATORY\_CS":2,"DATA\_TYPE":6,"CODELIST\_ID":7, "THIN\_REQUEST\_USAGE":16}

#disable/enable cms module. If module is disabled, no data regarding cms should be sent in response, otherwise exception might be thrown.

eai.cms.enabled=true

# for dam

#settings have the same meaning as for cms

eai.dam.communication.uri=http://localhost:12000/partner

eai.dam.communication.services={"search":{"GET":{"uri":"/media/images.json", "timeout":72000}},"retrieve":{"GET":{"uri":"/media/{0}/images/{1}.json", "timeout":-1}},"content":{"GET":{"uri":"/media/{0}/images/{1}", "timeout":-1}},"logging":{"POST":{"uri":"service/log", "timeout":72000}}}

eai.dam.communication.security.trust.all.certificates=true

eai.dam.communication.security.token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

eai.dam.config.model.xlsx.path=N:/wildfly-9/config/eia/dam

eai.dam.config.model.xlsx.mapping={"TITLE":0,"URI":12,"PPROPERTY\_ID":13,"MAPPING\_DATA\_CONVERT":14,"MAPPING\_CRITERIA\_CONVERT":15,"MANDATORY\_CS":2,"DATA\_TYPE":6,"CODELIST\_ID":7, "THIN\_REQUEST\_USAGE":16}

eai.dam.enabled=true

## Questions and Comments Raised by Technical POCs with Responses

Q: Yale has multiple CMS (and DAM?) systems. How is that accounted for in the design?

A: We have added a mandatory “source” property to the base image and object ontology definitions as well as to the Service URLs for acquiring the full records of images and objects and the content of images. In this way, it is only still necessary to register one set of service end points with ConservationSpace while providing for the possibility of multiple CMS and DAM sources.

Q: Suggestion made to use “media” instead of “image” for namespace.

A: Since attributes like “width” and “height” are specific to certain classes of media assets like images and video, Sirma would have to implement support for namespace inheritance, e.g. image and video extends media, and potentially support two or more name spaces for media items. The current approach is **not incompatible** with namespace inheritance, so we feel it is safe to continue limiting the media namespaces to “image” for now and to later pivot to using a combination of “media” and namespaces for other media types in the future if / when Sirma builds support for namespace inheritance. That said, we **have** modified the API to rename the image search, record, and content services to “media” services. This means the “media type” is now an important part of the URL for such services and the model can be extended in the future to support additional types of media assets.

Q: Is there a plan to provide some example implementations or to release a generic API solution that can be implemented by Partners to provide the desired functionality?

A: The NGA will release all of its code for the full implementation to GitHub when integration tests are successful. However, the NGA’s implementation is based on an in-memory cache of the approximately 200,000 cultural objects defined in our CMS to provide the CMS portion of the solution and a hybrid approach (cache + database queries) to provide the DAM solution. One could certainly provide an implementation of the interfaces we’re using, but the overall architecture of the solution is somewhat dependent upon having a collection data set that can fit in about 4GB of RAM. The approach would not scale to collections with millions of objects and might be overkill for collections with only a few thousand objects. Still, the basic interfaces and approach to handling the services can be used as a model for one possible solution to building an API implementation and in that sense, having access to working code would be valuable. The NGA solution is built upon Spring Rest and Spring Boot which itself, provides a good lift in productivity for partners with development staff who are familiar with Java.

Q: Could thumbnails work from a IIIF server?

A: Absolutely, regardless of whether they are URLs or read by the implementation and encoded in base64. The NGA intends to use this in order to provide thumbnails perfectly sized to fit within the 100 pixels of the application.

Q: Can we have a poll of hat LMS/CMS systems partners are using?

A: Yes – getting answers to such questions can be facilitated by the ConservationSpace technical google group. In fact, I will copy and paste all of these questions and answers there.

Q: Does this (presumably the response of cultural object and image records) involve creation of a generic “record” that transcends ConservationSpace’s needs?

A: We wanted to design these APIs so they would be useful outside of the context of ConservationSpace which is why they are so simple and extendable. There are really only a handful of fields actually required by ConservationSpace from all partners and extraneous fields are ignored by ConservationSpace so we hope the API design is flexible enough to power other systems once implemented. Since Sirma is well underway with implementation of this design, changing the overall approach would be a bit of work at this point, but if there are suggestions for alternative generic “record” formats, please post to the Google Group so we can talk about the merits of the various options, even if the discussion has to be limited to R3 timeframes.

Q: It would be great if CS could display images from external IIIF servers.

A: Actually, it can when Mirador is used in “exploratory” mode, i.e. its configuration hasn’t been locked by the report author, by entering the Manifest URL in the “add object” window. The URL won’t be saved in ConservationSpace in R2, but we could certainly consider passing IIIF resource URLs with the object data in R3 or persisting the Manifest URLs entered into Mirador. However, if a partner institutions adopts such practices, there is a probability that increases with time, that the images used for reports in ConservationSpace would no longer be available. So, this would have to be a risk the institution is willing to take; that is, relying on resources outside of ConservationSpace to render documents archived within ConservationSpace. It is this risk that is the basis for making a copy of all images and data that are imported or uploaded into the system. These documents are supposed to be available forever.

Q: Can you do free text searches against object descriptions and / or comments fields?

A: In R2, the CMS and DAM interfaces have a limited number of fields that can be searched – a partner could certainly override the intended behavior by searching outside of the recommend scope of the search, but it’s probably better to discuss this need within the partnership first and to extend the options for searching in R3 after conservators have gotten a feel for what’s working well and working poorly with respect to locating CMS and DAM objects and images.

Q: We have 30 paintings with the same title, how would that work?

A: If you entered the title, you’d get 30 results and would have to select the proper object based on information contained within the “search header” template defined for your institution. Many searches will return multiple objects or images. A search for titles with the letter “e”, for example, returns over 100,000 objects in the NGA collection. Obviously the key is to enter more specific search criteria to reduce this number, but the presence of potentially large result sets is why we introduced pagination into the API.

Q: What will be the procedure for importing multiple objects and images?

A: Execute a search, click on individual boxes of objects / images (I think there’s a select all in the interface as well), and click the “import/update” button.

Q: Are “references” mandatory on cultural object records?

A: If there are such references, yes, and the related works will be displayed in the search header with a clickable object number of the object representing the “hit” in the search. The implications of having no references is that it might become more difficult to locate the specific object record of interest or to import all of the child objects comprising a parent object.

## API Release Notes

V2.4.4

* Added Q&A section to capture suggestions, concerns, and questions about this document.
* Added mandatory sources property to all records and their service URLs.
* Added capability for service caller to suppress thumbnails and references when not needed
* Clarified that the current version of images may be updated manually on-demand and the current version of primary depictions of objects will be updated automatically by ConservationSpace when the object data is refreshed
* JSON suffix added to services that previously had no extension
* Various adjustments to the expected response formats, parameter handling, and validation
* Changed technique to be used for conveying the desired sort order of result sets
* Clarification provided for certain features that may be optionally implemented but will not be exercised by ConservationSpace in R2
* Solicited and incorporated feedback from 3/10/2016 technical POC call
* Updated image related search, record, and content services to reference “media” rather than “image” although “images” shall be the only required implementation for ConservationSpace R2.