API Technical Guide

This document provides user documentation for the Symplectic Elements API version 4.6 and provides your technical staff with the ability to fully integrate staff research data in either direction with any of the systems at your institution.

SYMPLECTIC

Symplectic Elements 4.6 - API Technical Guide

Contents

Readme

Together with the accompanying XML schema, this document provides user documentation for the Symplectic Elements API version 4.6 and provides your technical staff with the ability to fully integrate staff research data in either direction with any of the systems at your institution.

By default, the API is implemented as a secure REST style web service serving ATOM feed documents over HTTPS at an endpoint address defined by your institution.

Much of the human readable documentation for the details of request and response content is contained within the accompanying XML schema document. Don't forget to consult this document when interpreting response content or crafting requests.

For the examples in this documentation, the endpoint address, where used, will show as https://localhost:8091/elements-api, though of course at your institution the endpoint address will be different, particularly if your system administrator has exposed an unsecured endpoint. All relative URIs in this document will be relative to this endpoint base address.

When viewing the output of the API using a web browser, you will not see all of the returned data. You will typically just see a concise summary of the data, with many details invisible to you.

This is by design. The full data content of API responses is hidden within the ATOM feed itself using XML not typically shown by browsers. In order to view all the data in an API response, use your browser's "View Source" functionality.

The API is designed to allow your institution to implement a nightly user feed to keep the list of users of the system up-to-date, to allow you to implement a regular feed of research data from any of your systems into the Elements System, and to allow your external systems to search for and cache comprehensive and verified research data for their own purposes.

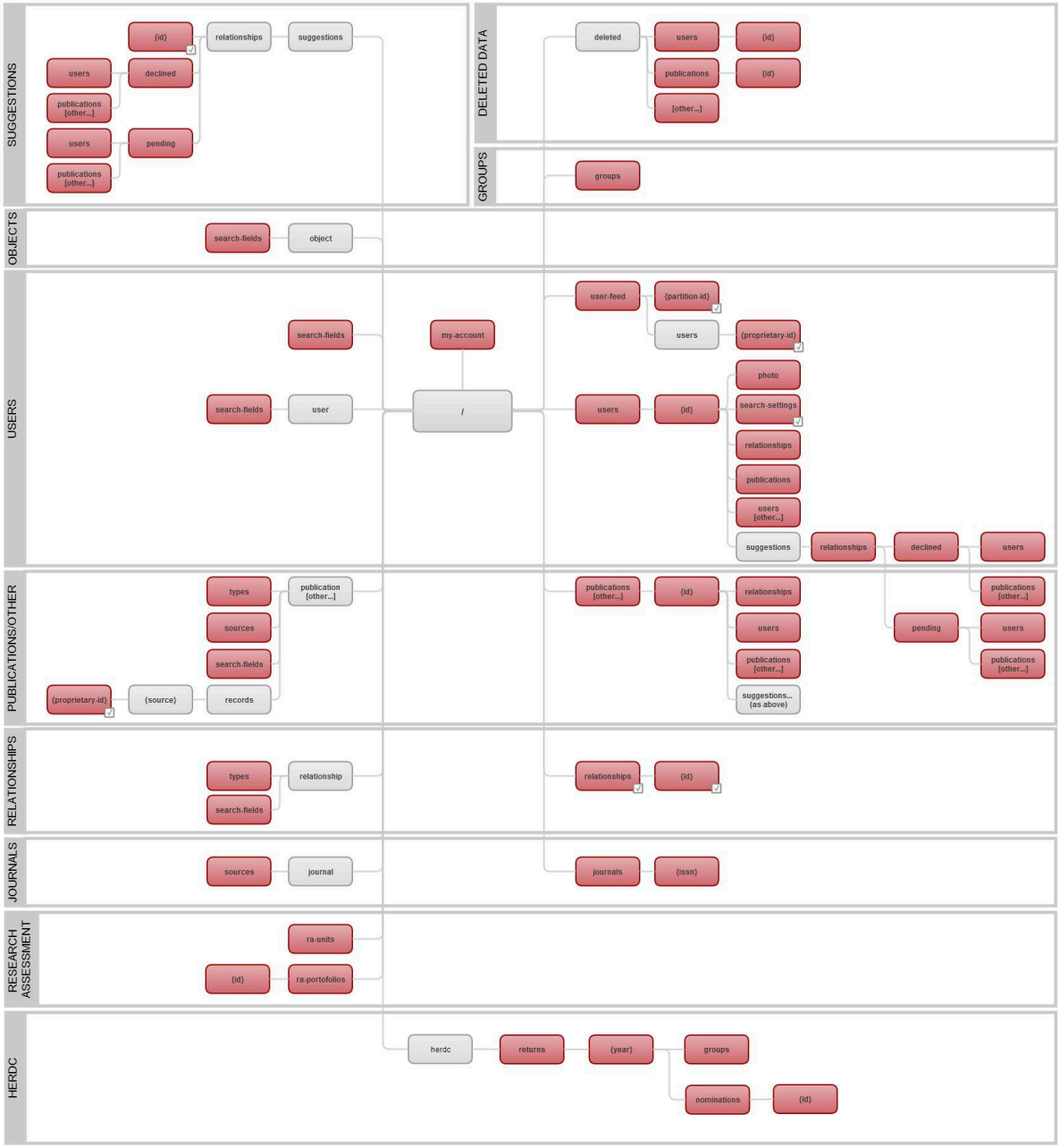
Such data can for example then be served by your institution's professional web page system, making your research data constantly available to the world.

Some workflow operations are also supported by the API, for clients who wish to manage the Elements system's powerful automated search capabilities from external systems.

The API is not designed to expose the Elements System as a high throughput real-time data server in a performance-critical role (for example, real-time provision of page data as part of a highly responsive web page system). To achieve this goal, implement an appropriate caching solution in front of the Elements system.

QUICK START

API site map



Red path elements represent resources. Grey path elements merely form a part of the path that represents more deeply nested resources. Resources with a tick icon indicate support of data modification operations (POST, PUT or PATCH based operations).

For brevity, 5 structurally identical sections of resources have been omitted and are together represented by the "PUBLICATIONS/OTHER" section. These sections are: professional activities, grants, projects, teaching activities, and organisational structures. See the reference section for their exact URLs.

Quick Examples

Intention	Operation
List all user IDs	GET /users
List all users	GET /users?detail=full
Get a particular user by ID	GET /users/8
Get a particular user by proprietary ID (HR identifier)	GET /users/pid-A77865443
Get a particular user by username	GET /users/username-jdjones
List all publications authored by a user	GET /users/5/publications?types=8&detail=full
List the IDs of all publications related to "blood flow"	GET /publications?query='blood flow'
List the IDs of all pieces of equipment funded by a particular grant	GET /grant/1089/equipment?types=16
List all projects related to a particular department	GET /projects?groups = 45
List the IDs of all users whose profiles have been updated since a given time	GET /users?modified-since=2012-06-21T13%3A34%3A00Z
Get the third page of the list of all publication IDs	GET /publications?page=3
List the pending publication suggestions for a particular user	GET /users/username-jdjones/suggestions/relationships/pending/publications

GUIDE

Coordination

By default, the API is configured to reject calls from unauthenticated clients. Authentication is implemented using the simple HTTP Basic authentication method (over a secure TLS channel), and authorisation to view or modify data is granted according to the rights associated with your credentials. Each and every call to the API needs to submit valid credentials in the appropriate way.

In addition, the API is configured to reject calls from all but a specified list of allowed client IP addresses, so you should contact the system administrator at your institution to arrange for appropriate access to the API for your system, by registering your IP address and finding out and using your credentials during requests.

Without properly authenticating, you will only receive HTTP 401 Unauthorised responses.

Alternatively, your system administrator may choose to expose an unsecured API endpoint. In this case, authentication will not be required (and indeed, you should avoid supplying credentials to an unsecured endpoint, as they will be sent in the clear via HTTP).

You must also arrange with the system administrator the manner of your planned use of the API, as they may for example wish to have you avoid placing undue stress upon the system during office hours, when the Elements database is typically required to be responsive for its user-interface.

Other considerations may include avoidance of coincidence with database backup schedules and heavy usage of the Elements database by other systems.

When writing code to consume the API, do not make calls so frequent that the Elements System becomes the bottleneck for speed of data-transfer.

If your program is pulling data as quickly as it can, then it is placing too heavy a burden on the Elements System. A simple policy of regular breaks in processing will allow the system time to recover from heavy queries. Make your application sleep between receiving a response and issuing its next request. A starting value of 500ms may be appropriate.

Security

Any request to a secure API endpoint that does not supply authentication credentials will return a response with a 401 (Unauthorized) code and empty content, as well as a WWW-Authenticate header prompting you to reissue your request with **HTTP basic access authentication** credentials.

For the interested, see http://en.wikipedia.org/wiki/Basic_access_authentication for a summary of HTTP basic access authentication.

The 401 response and WWW-Authenticate response header sent back by the API are well understood by browsers, which in turn will offer the user a login dialog before reissuing the original request for you with the supplied credentials attached. Your browser will then usually remember your credentials and resubmit them for you on each subsequent request while you browse the API's resources.

In the case of your programmatic client however, you must submit your credentials (username and password) yourself by supplying your own "Authorization" request header with a value generated by:

i) Using UTF-8, encoding to a byte array the text consisting of the username followed by a colon followed by the password;
ii) Converting the byte array to its base 64 text encoding;
iii) Appending the base 64 encoded credentials to the text "Basic " (note the trailing space).

This authentication does not begin a session and does not return any cookies to your client. You are simply required to provide your credentials anew on every request.

If your credentials are for some reason invalid, you will receive a 401 (Unauthorised) response.

A successfully authenticated request may also return a 403 (Forbidden) response, if your account does not have the rights to perform the operation you are attempting.

To see the permissions associated with your account, use the /my-account resource.

If you have problems authenticating, see the Troubleshooting section.

Please note that although passwords are sent "in cleartext", in the normal running of the system they are sent only over a secure connection (HTTPS), and so are entirely safely transmitted.

Note: if your system administrator has chosen to expose an unsecured API endpoint, this can be accessed without authentication. You will never receive a 401 (Unauthorized) response, and will have full rights to perform all operations.

When accessing the /my-account resource on an unsecured endpoint, you will see the details for an anonymous account.

You **must not** send a username and password to an unsecured endpoint: they will be sent in cleartext over a standard HTTP connection, and will **not** be secure.

Backwards compatibility

Symplectic Elements provides a simple mechanism for you to access older versions of the API in order to allow you a period of grace during which your legacy client can continue to work with the system while you perform upgrade work.

The old v3.7, and v3.7.16 APIs can be served from the latest version of Elements if your administrator so wishes.

For example, to continue to use the v3.7.16 API functionality, contact your system administrator and ask for the legacy API base URL for that endpoint. This might be http://localhost:8090/publications-api/v3.7.16. Then you should work with the v3.7.16 API documentation (as opposed to this documentation) and the v3.7.16 endpoint.

You must re-point any clients written for interoperation with older versions of the API to the appropriate legacy endpoint for them to continue to use the system without needing to make any code changes. However, consuming a legacy endpoint will mean that you continue to have access only to legacy functionality. In some cases this means that new types or fields of data may be hidden, or appear slightly modified in order to be communicable while still respecting the older API's XML schema.

In addition, most legacy endpoints are not served over HTTPS, nor do they implement the newer authentication or authorisation model of the latest endpoints. Because your institution might require a tighter security model than those offered by the legacy endpoints, your system administrator may choose to disable the legacy endpoints, and should work with you to make sure you have upgraded your client before the legacy endpoints are disabled.

You must of course access the endpoint for the latest version of the API in order to perform operations introduced by the latest version of the system.

Walkthroughs

Implementing a feed to a website

As an example, consider a hypothetical implementation of a web-content data feed from the Elements System to your institution's web content management system.

The goal would be to show a summary of each of your institution's employee's publications through your institution's public-facing website.

To this end a simple client program would be written by you to consume the API. The client program would run perhaps once per night at a time agreed with the Elements system administrator to minimise disruption to other users of the Elements System.

The client program would get a list of all users in the system from the API, then loop through them to search for and obtain publication-lists for the users of the Elements System for whom an externally facing web-page exists.

The returned publication data would be cached by the client program in a place suitable for use by the content management system until the cache is refreshed the next time the client runs.

It would be possible to dynamically issue API requests to generate page content for the visitors of your site at the times it is required, but this approach is bad for a number of reasons.

It makes unnecessary use of network resources and of the Elements Systems itself, by re-fetching what is largely static data (on the timescale of hours or days) every time it is viewed.

To avoid exposing the Elements System to a potentially unlimited amount of usage, you should cache data such as this in your local system to isolate the Elements System from high volumes of repeated high priority data reads.

Although the API is reasonably architected to return data quickly, its purpose is not to act as a performance-critical search engine in a web environment.

The following short example shows a simple C# program designed to loop through all of the users in the Elements System and print all of their publication titles.

Note that the program can be written in any language that offers a web client programming model, such as Python, Ruby or Java, and can be modified to cache publication lists in your system.

```
//initialise useful variables
XNamespace ns = "http://www.symplectic.co.uk/publications/api";
int pageSize = 100;
int pageNumber = 0;
int pageResultsCount;

//loop through the pages of users
do {
```

```
   //update page variables
   pageResultsCount = 0;
   pageNumber++;
   string requestUrl = string.Format("https://localhost:8091/elements-api/users&per-
page={0}&page={1}", pageSize, pageNumber);

   //prepare the request
   HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUrl);
   request.Headers.Add(string.Format("Authorization: Basic {0}",
Convert.ToBase64String(Encoding.UTF8.GetBytes(string.Format("{0}:{1}", "username",
"password")))));

   //get the response
   HttpWebResponse response = (HttpWebResponse)request.GetResponse();
   Stream responseStream = response.GetResponseStream();
   XmlReader reader = XmlReader.Create(responseStream);
   while(reader.Read() && reader.NodeType != XmlNodeType.Element) ;
   XElement responseXml = (XElement)XNode.ReadFrom(reader);

   //get the user ids
   var userIDs = from apiObject in responseXml.Descendants(ns + "object")
   where apiObject.Attribute("category").Value == "user"
   select apiObject.Attribute("id").Value;

   //loop through the users on this user page
   foreach(var userID in userIDs) {
      //keep track of the number of users on this page of users
      pageResultsCount++;

      //initialise useful variables
      int pubsPageNumber = 0;
      int pubsPageResultsCount;

      //loop through the pages of publications
      do {
         //update page variables
         pubsPageResultsCount = 0;
         pubsPageNumber++;
         string pubsRequestUrl = string.Format("https://localhost:8091/elements-
api/users/{0}/publications?types=8&detail=full&per-page={1}&page={2}", userID, pageSize,
pubsPageNumber);
```

```
        //prepare a request to get the publications of the user
        HttpWebRequest pubsRequest = (HttpWebRequest)WebRequest.Create(pubsRequestUrl);
        pubsRequest.Headers.Add(string.Format("Authorization: Basic {0}",
Convert.ToBase64String(Encoding.UTF8.GetBytes(string.Format("{0}:{1}", "username",
"password")))));

        //get the response
        HttpWebResponse pubsResponse = (HttpWebResponse)pubsRequest.GetResponse();
        Stream pubsResponseStream = pubsResponse.GetResponseStream();
        XmlReader pubsReader = XmlReader.Create(pubsResponseStream);
        while(pubsReader.Read() && pubsReader.NodeType != XmlNodeType.Element) ;
        XElement pubsResponseXml = (XElement)XNode.ReadFrom(pubsReader);

        //get the publication titles for the publications on this publications page
        var titles = from publication in pubsResponseXml.Descendants(ns + "object")
           let title = publication.Descendants(ns + "field").Where(f =>
f.Attribute("name").Value == "title").First()
           select title.Element(ns + "text").Value;

        //print them to screen
        foreach(string title in titles) {
           Console.WriteLine(title);
           pubsPageResultsCount++;
        }
     } while(pubsPageResultsCount == pageSize);
  }
} while(pageResultsCount == pageSize);
```

Please note that the listing above is an example only, and does not contain all the code required of a production version of a data-fetch program.

You should include error handling, logging, breaks in processing to prevent overloading of the server, robustness against server response failure, null reference checking and other things generally required of a reliable program.

When implementing your own web feed that displays publication information (or any research information) in the context of particular users, **do not forget to test and take heed of the "is-visible" property of each related object (see the API schema documentation for more information).**

This information is there for a very important reason and provides the only mechanism available for the user to indicate when they do not wish certain relationships to be made publicly viewable.

This can be an important issue for researchers publishing in the area of animal experimentation or other sensitive areas of research.

Implementing a user import feed

There are two alternative approaches to the implementation of a user feed from your HR system to the Elements System.

1. The first method is to use the user feed partition operations to first clear the user feed table (or the partition thereof that you have been assigned) and then re-upload in bulk your user feed entries. These operations are the DELETE /user-feeds/{partition-id} and POST <import-users-request/> /user-feeds/{partition-id} operations.
2. The second method is to add, update and delete user feed entries on a user-by-user basis using the more fine-grained user feed control operations PUT <user-feed-entry/> /user-feed/users/{proprietary-id} and DELETE /user-feed/users/{proprietary-id}.

To use the bulk update approach, you need to be assigned a partition ID, which you will use to identify the part of the user feed that you are in control of. Please contact the Elements system administrator at your institution to obtain a partition ID for the part of the user feed that you have been assigned.

To use the individual-user approach, you do not need any partition ID.

If using the bulk-update approach, you must then operate only on the user feed partition assigned to you in all of your calls to the user feed partition operations.

Whichever method you choose, the user feed works internally by storing the details of the user entries that you upload using these operations in a holding table (the user feed table), until such time as the Elements System administrator has configured the user feed data to be processed (typically once a night).

Only once the user feed table has been processed by the system will data-changes to the users in the Elements System become apparent.

When using the bulk-update approach, you should use the POST <import-users-request/> /user-feeds/{partition-id} operation to add new entries to the user feed table, and use the DELETE /user-feeds/{partition-id} operation to remove all entries in the user feed table previously added by you.

In this way you are in complete control of the records in the user feed table associated with your partition ID. You will not affect entries associated with other partition IDs so long as you only operate on your own assigned feed resource /user-feeds/{partition-id}. If you are one of many bulk-update user feed providers, you will then not affect the user data uploaded by other user feed providers in your institution who use a different partition ID.

When using the individual-user approach, you have fine-grained control over the whole user feed table and must be careful not to delete or modify user-entries uploaded by other feed providers in your institution.

At the scheduled time, the system compares the full list of entries in the user feed table with the list of users of the system, and reconciles the two lists by adding new users to the system for records in the user feed table not matching existing users of the system, by deactivating previously feed-imported users of the system now without a corresponding record in the user feed table, and by updating the user details for existing users with a corresponding record in the user feed table.

The correspondence of users previously uploaded by calls to this operation is achieved by comparing the Proprietary IDs of users with the Proprietary IDs in the user feed table provided in calls to the user feed operations.

The Proprietary ID you provide for each user **must therefore be unique** amongst all users that have ever been fed to the system, and **remain unchanged** for each user.

In this way, you (and other user feed providers) are the authority for Proprietary ID values in the Elements System for all the users uploaded using these operations. It is up to you and the Elements system administrator to enforce uniqueness and persistence.

You must work with the Elements system administrator to make sure that Proprietary IDs fed to the Elements System are unique across all data uploaded by the various user feed providers.

If you are the only user feed provider, you must simply make sure that the Proprietary IDs you provide are unique, that they remain unchanged, and that they are not recycled.

If you are not the only user feed provider, you must work with the Elements system administrator to additionally make sure that there can be no possible clashes between the Proprietary IDs provided by you, and the Proprietary IDs provided by other user feed providers.

Once the user feed table has been processed by the system, its data is not discarded. Rather it is kept, and if no changes are subsequently made to the table by your calls to the user feed operations, then the same data is re-processed the next time the user feed table is processed.

Because the user feed data is kept, if you wish to provide a full and fresh feed of all your users using the bulk-update approach, you must begin by making a call to the DELETE /user-feeds/{partition-id} operation, so that when you upload users with the POST <import-users-request/> /user-feeds/{partition-id} operation, you are not adding duplicate entries to the user feed table.

This clearing and updating requirement does not apply if you use the individual-user update approach.

As an aside, the deactivation of a user from the Elements System by omission from a user feed is not such a drastic occurrence as one might fear, because if the user later reappears in the user feed table, his/her records will be reactivated by the Elements System when the user feed table is next processed.

So do not panic if you accidentally deactivate a swathe of users for a day. The next time you include the deactivated users in your feed to the system, they will reappear in the Elements System with their research data fully intact.

To provide the Elements System with a user feed, you should write a client program that runs regularly, perhaps once a night, to inform the Elements System of the current list of users at your institution (when using the bulk update approach), or to inform the Elements System of any changes to individual users (when using the individual-users approach).

You must arrange with the Elements system administrator the timing of your calls to the user feed operations, in order not to interfere with conflicting operations scheduled for execution in the system, such as the processing of the user feed table itself.

```
<import-users-request xmlns="http://www.symplectic.co.uk/publications/api">
  <users>
```

```
<user>
  <title>Dr</title>
  <initials>JD</initials>
  <first-name>June</first-name>
  <last-name>Jones</last-name>
  <known-as></known-as>
  <suffix>FRS</suffix>
  <email>somebody@somewhere.org</email>
  <authenticating-authority>IC</authenticating-authority>
  <username>jonesjd</username>
  <proprietary-id>AA1229582</proprietary-id>
  <primary-group-descriptor>physics</primary-group-descriptor>
  <is-academic>true</is-academic>
  <is-login-allowed>true</is-login-allowed>
  <is-current-staff>true</is-current-staff>
  <arrive-date>2009-02-03</arrive-date>
  <generic-field-10>0123456789</generic-field-10>
</user>
<user>
  <title>Mr</title>
  <initials>TW</initials>
  <first-name>Terence</first-name>
  <last-name>Smith</last-name>
  <known-as>Terry</known-as>
  <suffix></suffix>
  <email>somebody@somewhere.org</email>
  <authenticating-authority>IC</authenticating-authority>
  <username>smithtw</username>
  <proprietary-id>GH8234623</proprietary-id>
  <primary-group-descriptor>mathematics</primary-group-descriptor>
  <is-academic>true</is-academic>
  <is-login-allowed>false</is-login-allowed>
  <is-current-staff>false</is-current-staff>
  <arrive-date>2004-02-03</arrive-date>
  <leave-date>2009-10-05</leave-date>
  <generic-field-10>0123456789</generic-field-10>
```

```
    </user>
  </users>
</import-users-request>
```

The XML above shows an example of the XML document you would supply when adding two hypothetical users to the user feed table using the POST <import-users-request/> /user-feeds/{partition-id} operation. The same two users could be supplied by two separate calls to the PUT <user-feed-entry/> /user-feed/users/{proprietary-id} operation, in which case for example the first call would require you to supply the following document:

```
<user-feed-entry xmlns="http://www.symplectic.co.uk/publications/api">
  <title>Dr</title>
  <initials>JD</initials>
  <first-name>June</first-name>
  <last-name>Jones</last-name>
  <known-as></known-as>
  <suffix>FRS</suffix>
  <email>somebody@somewhere.org</email>
  <authenticating-authority>IC</authenticating-authority>
  <username>jonesjd</username>
  <proprietary-id>AA1229582</proprietary-id>
  <primary-group-descriptor>physics</primary-group-descriptor>
  <is-academic>true</is-academic>
  <is-login-allowed>true</is-login-allowed>
  <is-current-staff>true</is-current-staff>
  <arrive-date>2009-02-03</arrive-date>
  <generic-field-10>0123456789</generic-field-10>
</user-feed-entry>
```

Note that the "user" and "user-feed-entry" elements from the two examples above, though named differently, are both of the same XSD element type as defined in the API schema. This type is sensitive to the order in which elements appear. Supplying elements in the wrong order will cause data not to be read, or other errors.

We will explain some of the elements available for user feed entries here. See the API XML schema for additional help.

```
<known-as>Terry</known-as>
```

The “known-as” element allows you to provide an informal alternative to the first-name of the user. For example, for a user whose “first-name” is “Jonathan”, you might provide a “known-as” value of “Jon”.

```
<suffix>FRS</suffix>
```

Use the “suffix” element to provide any official awards or society memberships that might appear as letters after the user’s surname. For example: “CBE FRS”.

```
<authenticating-authority>IC</authenticating-authority>
<username>jonesjd</username>
```

You must provide the authenticating authority ID and the username of the user.

This combination tells the Elements System which authentication system in your institution authenticates the user’s credentials when logging in to the Elements System, along with their associated username.

The authority/username combination **must be unique amongst all users currently supplied through the feed**, whether associated with a partition ID or not.

To get the list of available authenticating authority IDs, contact your Elements system administrator.

In the example above, the authenticating authority ID is “IC”. This identifies a system (possibly an LDAP server or a Single Sign-On scheme) to be used by the Elements System when “jonesjd” logs in using her username and password.

The available IDs are configured by the Elements system administrator, and each is associated internally with settings describing how the system is to interact with the authentication system in question.

```
<proprietary-id>AA1229582</proprietary-id>
```

You must provide the proprietary ID for the user. The proprietary ID **must be unique amongst all users ever supplied through the feed,** whether associated with your feed ID or not.

The proprietary ID is the only information used by the Elements System to identify the records of previously uploaded users with the records you are now supplying.

If you are supplying the user-entry data using the PUT <user-feed-entry/> /user-feed/users/{proprietary-id} operation, the URL to which you PUT the user's data must contain the same proprietary ID as is written in the XML data you provide here in the body of the request.

For example, for this user, the URL for this operation would have to be /user-feed/users/AA1229582

```
<primary-group-descriptor>physics</primary-group-descriptor>
```

Each user is a member of exactly one primary group. If the primary group descriptor is not supplied, the user’s primary group will be the single built-in top-level group, which contains all users.

Please liaise with your Elements system administrator to get the descriptors of the primary groups.

The primary group of a user determines which administrative part of the Elements System the user will belong to and consequently many of the application settings the user will see when they log in.

```
<is-academic>true</is-academic>
```

The Elements System allows most users to manage their own list of publications and other research data, and can be configured to search online databases for publications and other research data associated with those users.

However, some users of the system are not publishers of research material, and should have search facilities along with the ability to manage their own research data disabled.

This is typical of users who play an administrative role in the Elements System. For these users, specify “is-academic” to be “false”. For users who should have their own research data, specify “true”.

```
<is-login-allowed>true</is-login-allowed>
```

Specify "true" if the user should be allowed to log in to the system, and "false" otherwise.

```
<is-current-staff>true</is-current-staff>
```

Specify "true" if the user is currently a member of staff at your institution, or false if the user has left or has not yet arrived.

```
<arrive-date>2004-02-03</arrive-date>
<leave-date>2009-10-05</leave-date>
```

If appropriate and known, supply the dates on which the user arrived as a member of staff at your institution, and the date on which they left.

```
<generic-field-10>0123456789</generic-field-10>
```

The Elements system administrator may have configured the system to use a number of “generic” data fields for the users of the system.

Please liaise with the system administrator to arrange the injection of appropriate data for these fields.

Please note that the generic fields numbered 11 and above are defined to be sensitive HR data fields. Data placed in these fields is subject to tighter security when accessed through a secure API endpoint, for example. See the Security section for more details.

In this example, a phone number has been supplied on the understanding that field number 10 always holds the office phone number of a user.

The following C# code was used to create the example POST <import-users-request/> /user-feeds/{partition-id} request, above. You can modify this code to write a client that provides a regular user feed to the Elements System using the bulk-update approach.

```
//declare useful variables - let's assume we have been
//assigned the user feed partition with id "hr"
string requestUrl = "https://localhost:8091/elements-api/user-feeds/hr";
XNamespace ns = "http://www.symplectic.co.uk/publications/api";

//declare two users to be fed to the Elements System
var users = new[] {
   new {
      Title = "Dr", Initials = "JD", FirstName = "June",
      LastName = "Jones", KnownAs = "", Suffix = "FRS",
      Email = "somebody@somewhere.org",
      Authority = "IC",
      Username = "jonesjd", ProprietaryID = "AA1229582",
      PrimaryGroupDescriptor = "physics",
      InternalPhoneNumber = "0123456789",
      LoginAllowed = true,
      IsCurrentStaff = true,
      ArriveDate = new DateTime(2009, 2, 3),
      LeaveDate = (DateTime?)null
   },
   new {
      Title = "Mr", Initials = "TW", FirstName = "Terence",
      LastName = "Smith", KnownAs = "Terry", Suffix = "",
      Email = "somebody@somewhere.org",
      Authority = "IC",
      Username = "smithtw", ProprietaryID = "GH8234623",
      PrimaryGroupDescriptor = "mathematics",
      InternalPhoneNumber = "0123456789",
      LoginAllowed = false,
      IsCurrentStaff = false,
      ArriveDate = new DateTime(2004, 2, 3),
      LeaveDate = (DateTime?)new DateTime(2009, 10, 5)
   }
};

//prepare the request
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUrl);
```

```
request.Headers.Add(string.Format("Authorization: Basic {0}",
Convert.ToBase64String(Encoding.UTF8.GetBytes(string.Format("{0}:{1}", "username",
"password")))));
request.Method = "POST";
request.ContentType = "text/xml";
Stream body = request.GetRequestStream();
using(XmlWriter xmlWriter = XmlWriter.Create(body)) {
   XElement requestXml = new XElement(ns + "import-users-request",
   new XElement(ns + "users",
   from user in users
   select new XElement(ns + "user",
      new XElement(ns + "title", user.Title),
      new XElement(ns + "initials", user.Initials),
      new XElement(ns + "first-name", user.FirstName),
      new XElement(ns + "last-name", user.LastName),
      new XElement(ns + "known-as", user.KnownAs),
      new XElement(ns + "suffix", user.Suffix),
      new XElement(ns + "email", user.Email),
      new XElement(ns + "authenticating-authority", user.Authority),
      new XElement(ns + "username", user.Username),
      new XElement(ns + "proprietary-id", user.ProprietaryID),
      new XElement(ns + "primary-group-descriptor", user.PrimaryGroupDescriptor),
      new XElement(ns + "is-academic", true),
      new XElement(ns + "is-login-allowed", user.LoginAllowed),
      new XElement(ns + "is-current-staff", user.IsCurrentStaff),
      new XElement(ns + "arrive-date", user.ArriveDate.Value.ToString("yyyy-MM-dd")),
      //for null values, do not include the corresponding element
      user.LeaveDate == null ? null : new XElement(ns + "leave-date", user.LeaveDate
Value.ToString("yyyy-MM-dd")),
      new XElement(ns + "generic-field-10", user.InternalPhoneNumber))));

   requestXml.WriteTo(xmlWriter);
}

body.Close();

//get the response
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
StreamReader reader = new StreamReader(response.GetResponseStream());
string responseContent = reader.ReadToEnd();
```

The following C# code was used to create the example *create/update user in user feed* request, above. You can modify this code to write a client that provides a regular user feed to the Elements System using the individual-user update approach.

```
//declare some useful variables
string userProprietaryID = "AA1229582";
string requestUrl = string.Format("https://localhost:8091/elements-api/user-feed/users/{0}",
userProprietaryID);
XNamespace ns = "http://www.symplectic.co.uk/publications/api";


//declare a user to be fed to the Elements System
var user = new {
   Title = "Dr", Initials = "JD", FirstName = "June",
   LastName = "Jones", KnownAs = "", Suffix = "FRS",
   Email = "somebody@somewhere.org",
   Authority = "IC",
   Username = "jonesjd",
   ProprietaryID = userProprietaryID,
   PrimaryGroupDescriptor = "physics",
   InternalPhoneNumber = "0123456789",
   LoginAllowed = true,
   IsCurrentStaff = true,
   ArriveDate = new DateTime(2009, 2, 3)
};


//prepare the request
HttpWebRequest request = (HttpWebRequest)WebRequest.Create(requestUrl);
request.Headers.Add(string.Format("Authorization: Basic {0}",
Convert.ToBase64String(Encoding.UTF8.GetBytes(string.Format("{0}:{1}", "username",
"password")))));
request.Method = "PUT";
request.ContentType = "text/xml";
Stream body = request.GetRequestStream();
using(XmlWriter xmlWriter = XmlWriter.Create(body)) {
   XElement requestXml = new XElement(ns + "user",
      new XElement(ns + "title", user.Title),
      new XElement(ns + "initials", user.Initials),
      new XElement(ns + "first-name", user.FirstName),
      new XElement(ns + "last-name", user.LastName),
      new XElement(ns + "known-as", user.KnownAs),
      new XElement(ns + "suffix", user.Suffix),
      new XElement(ns + "email", user.Email),
```

```
      new XElement(ns + "authenticating-authority", user.Authority),
      new XElement(ns + "username", user.Username),
      new XElement(ns + "proprietary-id", user.ProprietaryID),
      new XElement(ns + "primary-group-descriptor", user.PrimaryGroupDescriptor),
      new XElement(ns + "is-academic", true),
      new XElement(ns + "is-login-allowed", user.LoginAllowed),
      new XElement(ns + "is-current-staff", user.IsCurrentStaff),
      new XElement(ns + "arrive-date", user.ArriveDate.ToString("yyyy-MM-dd")),
      new XElement(ns + "generic-field-10", user.InternalPhoneNumber));
   requestXml.WriteTo(xmlWriter);
}
body.Close();

//get the response
HttpWebResponse response = (HttpWebResponse)request.GetResponse();
StreamReader reader = new StreamReader(response.GetResponseStream());
string responseContent = reader.ReadToEnd();
```

During the processing of the user feed table, the Elements System may discover duplicate entries for users (i.e. more than one entry with the same user proprietary ID), or find other problems with the data supplied by a feed.

These errors will be reported to the Elements system administrator, who may need to contact you to have you correct your feed.

Implementing a data import feed

The Symplectic Elements API provides operations that allow you to import your research data into the system in a manner that is designed to closely resemble the model used to import data from the various online data sources such as Scopus and PubMed.

Implementing such a data feed is relatively straightforward, though you should review the section Best Practices to help you to implement a reliable and easily debuggable data feed into the Elements System.

The current version of the import operations support both a one-off import and subsequent updates of activities, equipment, grants, organisational structures, projects and publications.

Additionally, the operations allow you to identify which users are to be associated with the data you have imported, and in what way (e.g. which users of the system are the primary investigators of the grants you have imported).

A typical example of usage of this functionality would be to implement a nightly feed from an authoritative grants database in your institution, keeping the Elements System up-to-date with the grants data managed by that system.

To remain consistent with the way data enters the system when external data sources such as the Web of Science are trawled, the broad approach taken to data import observes the following steps.

You are a data source

Your feeding system acts as if it were one of the external data sources registered within the Elements System.

This is simply a matter of you choosing the data source identifier for an appropriate data source already configured in the Elements System, and then supplying that identifier to the operations you use to upload data. All records uploaded through the API will then be associated with the indicated data source.

In order to prevent you from overwriting data that has come from another data source, not all data sources are open for use in this way. See the API schema for more information about data sources and records. The only data source initially configured for data import is the Institutional Grants System data source of the grants category.

Feed records into the system

You maintain control of the record data you feed from your data source, and the Elements System and its users collect your records into objects with any matching records from other data sources.

This approach works in exactly the same way as (for example) PubMed data and Scopus data already enter the system: PubMed and Scopus independently provide their publication records, and the Elements System and its users aggregate these where appropriate into the same publication object, without the interaction of PubMed or Scopus.

In this way, the Elements System and its users (and not the source feeding the data) remain in control of the deduplication of data, and the feeding system remains in control of the records it feeds into the system.

You can upload new records, and update previously uploaded records, using the PUT <import-record/> /{cat}/records/{source}/{proprietary-id} operation.

Feed associations to users into the system

You can also declare associations between two objects in the system that you have previously uploaded, using the POST <import-relationship/> /relationships operation.

Your associations are stored as relationships between two objects. Each association you provide will be interpreted and stored as a relationship between the objects you have identified.

Based on the association data you provide, the system will decide whether to create a new relationship, or update an existing relationship in order to model your association, always returning the representation of the relationship created or updated.

Update previously imported data

Subsequent import of the same record into the system will simply update the record, not create a new one. The record is identified by the combination of data-source and record proprietary ID that you specify for the record when importing it.

Subsequent import of the same association will achieve the same effect. The relevant relationship in the system will be updated. The relationship is identified by two object identifiers and the type of relationship you specify. Since the Elements system can hold a maximum of one relationship for any given combination of these three values, the correct relationship will be updated.

Deleting previously imported data

The API exposes operations that allow you to delete previously imported records and relationships. Once data is deleted, it is deleted, and is not recoverable except by restoring a complete database backup image.

It is strongly advised that you do not delete previously imported data, and that instead you fix any data problems using manual techniques. Discuss the situation carefully with your system administrator before committing to deleting data, whether as a one-off, or on a regular basis.

Nevertheless, you may wish to automate some deletions, and operations to allow you to achieve this are provided for completeness. If you do use API operations to delete records or relationships, you should make all the necessary data backups before commencing, and conduct a suitable review of data accuracy afterwards.

Walkthrough

This section walks you through a hypothetical implementation of a full data feed, upon which you can base your own data feeds.

The supposed situation is that of an independent grants system embedded in your institution (the "grants system"), from which a nightly data feed is to regularly update the Elements System with all of its grants data. This data includes both the grants and the relationships of those grants to the staff in your institution.

Equally, this walkthrough could be applied just as easily to an independent publications system, to feed publications data into the Elements system, or an independent projects and staff system, to feed project data into the Elements System.

The steps used for this implementation are:

1. Arrange access to the API
2. Select the data source that represents your system
3. Decide how you want to map your grants to Elements grants
4. Implement an importer program to import your grants
5. Decide how you want to associate users to your grants
6. Extend your importer program to import your associations
7. Liaise with the system administrator to decide when to run your importer

1: Arrange access to the API

Your importer program will run from a dedicated machine with a known IP address and using a dedicated API account with which to connect to the API.

You check with the system administrator that you have read/write access to the Elements System's API (by registering your machine's IP address with them and getting the credentials of an API account with read/write permissions on the Elements System).

2: Select the data source that represents your system

Using the /grant/sources resource, you see that of the grants data sources currently configured in the Elements System, the source with name "source-3" is an importable data source.

This means that you can use it to represent the grants system from which you will implement a data feed. This decision is agreed between you and the Elements system administrator and it is decided that this data source, not having been used to represent any other data, will represent the grants system.

Any data source not registered in the Elements system as an importable data source has effectively been locked from write operations through the API, protecting its data in the Elements system from being overwritten or added to by importers such as yourself.

3: Decide how you want to map your grants to Elements grants

You must decide how to translate a grant that exists in your grants system into a grant in the Elements System.

After performing a review of the grant data fields available in your grants system, and the grant types and their data fields currently configured in the Elements System (using the /grant/types resource), you must arrive at a desired mapping for converting your grant system's grants data to Elements grants data.

To keep things simple, let us suppose that all grants in your independent grants system will be mapped to the single grant type configured by your institution in the Elements System. Let us suppose that the /grant/types resource returns the following type data:

```
<api:type id="1" name="grant">
    <api:heading-singular>Research Grant</api:heading-singular>
    <api:heading-plural>Research Grants</api:heading-plural>
    <api:heading-lowercase-singular>research grant</api:heading-lowercase-singular>
    <api:heading-lowercase-plural>research grants</api:heading-lowercase-plural>
    <api:fields>
        <api:field>
            <api:name>funder-reference</api:name>
            <api:display-name>Funder reference</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>funder-name</api:name>
            <api:display-name>Funder name</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>title</api:name>
            <api:display-name>Title</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>description</api:name>
            <api:display-name>Description</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
```

```
        <api:field>
            <api:name>start-date</api:name>
            <api:display-name>Start date</api:display-name>
            <api:type>date</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>end-date</api:name>
            <api:display-name>End date</api:display-name>
            <api:type>date</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>amount</api:name>
            <api:display-name>Amount</api:display-name>
            <api:type>money</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>institution-reference</api:name>
            <api:display-name>Institution reference</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>application-date</api:name>
            <api:display-name>Application date</api:display-name>
            <api:type>date</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>award-date</api:name>
            <api:display-name>Award date</api:display-name>
            <api:type>date</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
```

```
            <api:name>status</api:name>
            <api:display-name>Status</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
        <api:field>
            <api:name>funder-type</api:name>
            <api:display-name>Funder type</api:display-name>
            <api:type>text</api:type>
            <api:is-mandatory>false</api:is-mandatory>
        </api:field>
    </api:fields>
</api:type>
```

There are plenty of fields available, but if you wish to store data in additional specific fields, you must arrange with your Elements system administrator to add the required fields to the desired grant types. We will assume that this has already been achieved.

Let us suppose that the grants data in your independent grants system belong to a very simple pair of database tables with the following schemas:

tblGrant

Column name	Column type	Properties
ID	int	primary key
Date	datetime	nullable
Value	int	nullable
Notes	varchar(200)	nullable

tblGrantStaff

Column name	Column type	Properties
Grant_ID	int	foreign key to tblGrant(ID)
Staff_ID	varchar(50)	not nullable

tblGrant holds the grants themselves, and tblGrantStaff holds a list of staff member associations to the grants in tblGrant, in a standard relational database way, where each association represents the "primary investigator" for a grant. The Staff_ID column it is assumed uses the same identifiers as

those stored as user proprietary IDs in the Elements System, and we assume that the data is clean, accurate and suitably free of duplicates.

If tblGrantStaff's Staff_ID column stored instead the equivalent of the Elements System's usernames instead of user proprietary IDs, you would still be able to import your data using usernames (see later).

Comparing these tables with the XML type information above, you decide on the following mapping:

tblGrant column name	Elements system grant field
ID	record proprietary ID
Date	start-date
Value	amount
Notes	<not to be mapped>

Note that the ID of the grant in your independent grant system is mapped to the Elements System record proprietary ID. Although the proprietary ID is not defined as a field for the grant type in the grant type XML, all records in the Elements System have proprietary IDs that represent the ID of the grant record as given to it by the data source from which it came, and so you **must** assign a proprietary ID to every record you import that can be uniquely traced back to the record from which it came in the external system.

The ID of the row from which the grant originally came in your grants system is assumed to be such an ideal identifier in this scenario, though it is up to you to appropriately choose which value is mapped to the record proprietary ID in the Elements system.

The proprietary ID that you assign when re-importing the same grant into the Elements System must remain unique and immutable, since the import of a grant with a changed proprietary ID would be considered the import of an entirely new grant by the Elements system.

4: Implement an importer program to import your grants

The operation to call is the PUT <import-record> /grant/records/source-3/{proprietary-id} operation. This operation will create a new record with the indicated proprietary ID if it doesn't already exist, or update the existing record with the data you supply if it does.

The following code snippet shows example code that imports two grants into the Elements system. The grants data is fixed, though you should instead implement a loop that takes the data you need from each grant in your grants system, and use that data instead.

```
//declare useful variables
XNamespace ns = "http://www.symplectic.co.uk/publications/api";


//declare two hypothetical grants to be fed to the Elements System
var grants = new[]
```

```
    {
        new
            {
                ProprietaryID = "0001234",
                Title = "The first grant",
                SterlingValue = "400000"
            },
        new
            {
                ProprietaryID = "0001235",
                Title = "The second grant",
                SterlingValue = "5500"
            }
    };


//for each grant, import the grant to the Elements System
foreach (var  grant in grants)
{
    string          requestUrl          =          string.Format("https://localhost:8091/elements-
api/v4.6/grant/records/source-3/{0}", Uri.EscapeDataString(grant.ProprietaryID));
    HttpWebRequest request = (HttpWebRequest) WebRequest.Create(requestUrl);
    request.Headers.Add(string.Format("Authorization: Basic {0}",
        Convert.ToBase64String(Encoding.UTF8.GetBytes(string.Format("{0}:{1}",        "username",
"password")))));
    request.Method = "PUT";
    request.ContentType = "text/xml";
    Stream body = request.GetRequestStream();
    using (XmlWriter xmlWriter = XmlWriter.Create(body))
    {
        XElement requestXml = new XElement(ns + "import-record",
            new XAttribute("type-id", 1),
            new XElement(ns + "native",
                new XElement(ns + "field", new XAttribute("name", "title"),
                    new XElement(ns + "text", grant.Title)),
                new XElement(ns + "field", new XAttribute("name", "amount"),
                    new    XElement(ns    +    "money",    new    XAttribute("iso-currency",    "GBP"),
grant.SterlingValue))));
        requestXml.WriteTo(xmlWriter);
    }
    body.Close();
```

```
    //get the response
    try
    {
        HttpWebResponse response = (HttpWebResponse) request.GetResponse();
        StreamReader reader = new StreamReader(response.GetResponseStream());
        string responseContent = reader.ReadToEnd();
    }
    catch (WebException doh)
    {
        string response = new StreamReader(doh.Response.GetResponseStream()).ReadToEnd();
    }
}
```

The code above calls the PUT <import-record> /grant/records/source-3/{proprietary-id} operation repeatedly, once for each grant to be imported, constructing the operation URL using the name of the source we chose earlier, the source from which we are importing ("source-3") and the proprietary ID of the grant record we are importing, as per the PUT <import-record> /grant/records/source-3/{proprietary-id} operation documentation.

The XML document forming the content of each HTTP request contains the rest of the data for the grant being imported by the request, and correct supply of this XML will constitute most of the effort of implementing the data feed.

The PUT <import-record> /grant/records/source-3/{proprietary-id} operation requires you to HTTP PUT an "api:import-record" element, as defined in the API schema. You can see this being done in the code example above. The schema definition for the element is:

```
<xs:element name="import-record">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="citation-count" type="xs:int" minOccurs="0"/>
      <xs:element name="verification-status" type="api:verification-status" minOccurs="0"/>
      <xs:element name="verification-comment" type="xs:string" minOccurs="0"/>
      <xs:element name="native" type="api:import-native-record"/>
    </xs:sequence>
    <xs:attribute name="type-id" type="xs:int" use="required"/>
  </xs:complexType>
</xs:element>
```

You must specify the type of record you are mapping to from amongst the types available in the relevant category (e.g. from /grant/types), as decided early in the mapping process for your import. In our hypothetical case, we decided earlier to map all grants to the only grant type currently configured

in the system, with type-id 1. When your institution makes more types are available, you can choose amongst them.

If you supply a citation-count, and the concept applies to the type of record you are importing (only publication records have citation counts), this value will be set against the record. If you do not supply a value, the citation count will not be updated.

You are also in control of your institution's verification status for this record. These concepts only apply to publication records, and values supplied when importing other categories of record will be ignored. If you do not supply a verification status value, the system may set the verification status to "unverified" in any case, since you have modified the record's data. The verification comment is updated only when you supply a verification status. If in this case you do not supply a verification comment, then any existing comment is deleted.

The next element you must supply is the "native" element, containing all the basic field values for the record you are importing. The format of most of this element will be familiar to you from when you take data from the API, as it is just a simplified version of the "api:native" element you see when exporting objects from the API. See the API schema for more detailed help.

```
<xs:complexType name="import-native-record">
    <xs:sequence>
        <xs:element name="field" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:choice>
                    <xs:element name="addresses">
                        <xs:complexType>
                            <xs:sequence>
                                <xs:element name="address" type="api:address" minOccurs="0" maxOccurs="unbounded"/>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="boolean" type="xs:boolean">
                    </xs:element>
                    <xs:element name="date" type="api:date">
                    </xs:element>
                    <xs:element name="decimal" type="xs:decimal">
                    </xs:element>
                    <xs:element name="funding-acknowledgements" type="api:funding-acknowledgements">
                    </xs:element>
                    <xs:element name="identifiers">
                        <xs:complexType>
                            <xs:sequence>
```

```
                        <xs:element name="identifier" type="api:identifier" minOccurs="0"
maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="integer" type="xs:int">
            </xs:element>
            <xs:element name="items">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="item" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="keywords">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="keyword" type="api:keyword" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="money" type="api:money">
            </xs:element>
            <xs:element name="pagination" type="api:pagination">
            </xs:element>
            <xs:element name="people">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="person" type="api:person" minOccurs="0" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="text" type="xs:string">
            </xs:element>
        </xs:choice>
        <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
```

```
        </xs:element>
    </xs:sequence>
</xs:complexType>
```

Let's view an example of a full PUT <import-record> /grant/records/source-3/{proprietary-id} operation for the hypothetical situation we are in with our grants system. We assume that the ID of the grant record in tblGrant in your grants system is, say, "12", and that its StartDate is perhaps "2010-10-04" and its Value "3500", which we will presume is in UK Pounds (GBP).

The entire operation consists of an HTTP PUT to /grant/records/source-3/12 with the following content:

```
<import-record type-id="1" xmlns="http://www.symplectic.co.uk/publications/api">
  <native>
    <field name="start-date">
      <date>
        <day>4</day>
        <month>10</month>
        <year>2010</year>
      </date>
    </field>
    <field name="amount">
      <money iso-currency=" GBP" >3500</money>
    </field>
  </native>
</import-record>
```

And that's it for importing records. The code example introduced earlier creates a slightly different import document importing its data to different fields. You would construct your XML document to import to the fields according to the mapping you decided early on in the process.

5: Decide how you want to associate users to your imported grants

In our hypothetical situation, each row in tblGrantStaff represents an association between a member of your staff and one of the grants in your grants system. You will import these as user-relationships to the Elements system.

In this simplified situation, we assume that all of these associations represent the relationship of "staff member is the primary investigator on the grant", corresponding to the relationship type with ID 43 (see the GET /relationship/types operation and the API schema for more information).

At this stage, when implementing your own data feed, you will need to decide on a mapping between the associations in your grants system and the relationship types available in the Elements system.

6: Extend your importer program to import your associations

The operation to call is the POST <import-relationship> /relationships operation. This operation will create a new relationship between the record and user you indicate, of the type you indicate, if one doesn't already exist, or update the existing one with the data you supply if it does.

You will call the POST <import-relationship> /relationships operation repeatedly, once for each association between a grant and user to be imported, with the operation URL simply being /relationships,

The POST <import-relationship> /relationships operation requires you to HTTP POST an "api:import-relationship" element, as defined in the API schema. The schema definition for the element is:

```
<xs:element name="import-relationship">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="from-object" type="xs:string"/>
      <xs:element name="to-object" type="xs:string"/>
      <xs:choice>
        <xs:element name="type-id" type="xs:int"/>
        <xs:element name="type-name" type="xs:string"/>
      </xs:choice>
      <xs:element name="is-visible" minOccurs="0">
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:boolean">
              <xs:attribute name="overwrite" type="xs:boolean" use="optional"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

You must first identify the ‘from-object’ and the ‘to-object’ using api object identifiers (see page 64).

You must next specify the type of relationship being imported using the "api:type-id" or “api:type-name” element. This forms a part of the identity of the relationship itself, alongside the associated record and user. In our hypothetical situation, we have decided to map all associations in tblGrantStaff to Elements relationships of type ID 43 (user is primary investigator of grant).

The "api:is-visible" element is available for you to control the visibility status of the relationship between the user and the record. This concept does not apply to all relationship types and use of the element for an incompatible relationship type will cause an error. You must contact your system administrator to discuss which, if any, of the relationships you import must be flagged as invisible.

Not supplying the "api:is-visible " element will cause the Elements system to either set a standard default value for it (if creating the relationship for the first time), or leave the current value alone (if updating an existing relationship).

If you supply the "api:is-visible" element, you may also supply an "overwrite" attribute set to either "true" or "false". Not supplying the attribute will cause it to behave as if you had specified "false". When "overwrite" is set to "false", the value for "api:is-visible" that you supply will only be committed if the relationship is being created, but no changes will be made to it if the relationship already exists. Setting "overwrite" to "true" will cause the visibility status of the relationship to be overwritten with the value you supply in all circumstances.

Let's view an example of a full POST <import-relationship> /relationships operation for the hypothetical situation we are in with our grants system. We assume that the Grant_ID of the association row in tblGrantStaff in your grants system is, say, "12", and that the Staff_ID value is perhaps "000123", representing the proprietary ID of one of your members of staff.

The entire operation consists of an HTTP POST to /relationships with the following example content:

```
<import-relationship xmlns="http://www.symplectic.co.uk/publications/api">
  <from-object>grant(source=source-3,pid=001234)</from-object>
  <to-object>user(pid=000123)</to-object>
  <type-id>43</type-id>
  <is-visible>true</is-visible>
</import-relationship>
```

Please note that you are free to identify the user in the relationship using one of the three common identification schemes: by authority/username combination, by the Elements ID of the user, or by your institution's proprietary ID for the user.

7: Liaise with the system administrator to decide when to run your importer

Finally, you need to discuss and confirm with your Elements system administrator the times at which you run your importer.

Your system administrator should suggest a time that does not conflict with other heavy-duty operations being performed on the system. Data importers should typically be run at night time or weekends.

Best practices

When implementing a client to consume the API, observe the following best practices:

Record the Elements system version in your code comments

Keep a note of which version of the Elements System's API you wrote your code to interact with. Not doing this may cause significant problems when you, or others, revisit your code.

Log all requests and responses

Do not rely on the Elements system's logging alone. Some of your requests may not even reach the Elements system, for various reasons. Perhaps network issues are responsible, or the API has not been enabled. A bug in the API may prevent it from logging interactions, or the system administrator may have disabled certain levels of logging to ease the burden on the system.

Consider logging the following request information:

- Request URL and verb
- Exact time of request
- Size of content of request

Include an optional program parameter that will in addition cause your client to log the entire HTTP content of the requests it sends to the Elements system. You can switch this parameter on if you need to review the data you have been sending to the Elements system for debugging purposes, and switch it off again once you are done, to save disc space.

Consider logging the following response information:

- HTTP status code
- Any API error codes/messages returned in the body of the response
- The time taken from issuing the request to receiving the response
- All warnings returned by the API (look for an API warnings element in every ATOM entry element – see the section on Success Responses earlier in this document)

In addition, log any errors that occur in your own code.

Do not overload the Elements system

If you are in control of the timing and frequency of requests to the API, then build in a configurable wait time between the end of the processing of one request and the issuing of the next. Symplectic will ask you about how much sleep time you have built into your client between requests when

responding to Client Environment & Systems Support requests that enquire about performance issues.

If the Elements API is the bottleneck in the speed of data throughput, then you are overloading the Elements system and you should expect performance issues in either your system, or for other users and clients of the Elements system.

If you are not in control of the timing and frequency of requests to the API, then you have likely exposed the API indirectly to an uncontrolled and potentially high level of load, such as that which arrives in real-time through a web server environment. The API as a comprehensive data feed service is not designed to support such a role, and you should implement a caching solution to isolate the Elements system from frequent requests and unpredictable loads.

Use a caching solution where possible

To remove unnecessary points of failure in your distributed architecture, and to isolate the Elements system from an uncontrolled and potentially high level of load caused by (often repeated) requests for the same data arriving to the API at a variable and unpredictable rate, always use a caching solution where possible.

Search for data in the Elements database though the API regularly and cache the data in a separate system in the form most efficiently able to be served in your high/variable-load environment.

Symplectic will be happy to provide Client Environment & Systems Support in the form of advice or consultancy for implementations of such solutions as a part of your support agreement.

Not using a caching solution in a situation in which distributed load would otherwise be significantly lowered is not a supported use of Elements.

Validate your XML

When developing a client that supplies data to the API, make it a habit to explicitly check that any XML documents your client is submitting are both well-formed and validate against the API schema, using an XML validation tool. Such tools are available online free of charge. Some well-used tools for these purposes are http://validator.w3.org/ and http://www.xmlforasp.net/SchemaValidator.aspx, respectively.

Code defensively when reading response XML

Previously we have considered additions to existing enumerations in the API schema to be backwards breaking compatibility changes. For example, adding “teaching-activity” to the “api:category” type was considered to break compatibility, requiring a new endpoint. In the future we will no longer consider the addition of value to an enumeration as breaking compatibility. This will allow for quicker introduction of new functionality such as new relationship types and new categories.

Generate XML using XML libraries, not string formatting functions

Always use library code to generate your XML, and never generate it yourself using string-formatting and concatenation routines - this will always lead to trouble, since despite looking easy, XML is very hard to get right without using a proper XML library.

For example, beware that as per the XML standard, any Unicode characters not in the code point ranges 0x9, 0xA, 0xD, 0x20 - 0xD7FF, 0xE000 - 0xFFFD, 0x10000 - 0x10FFFF, if present in the content of your XML, render your XML invalid, whether included literally or character-referenced. These characters can easily appear in your documents if you have copied and pasted text from (for example) Microsoft Word, or if you embed data from a database of yours straight into an XML document as pure text.

All modern programming languages make available at least one good XML library.

Review your client once a year

Review the implementation of your client once a year, and bring its compatibility level up to date with the latest version of the API that you are running at your institution.

You might not have been aware that your institution has upgraded its Elements system, and that your client is still using an old compatibility endpoint that will one day stop being supported, so scheduling a regular review of your client is a good idea. Contact your system administrator and ask which version of the system is running, compare this with the version you wrote your client to work with, and schedule time to upgrade your client.

This way, you will not be faced with having to suddenly update an old client written against a very old API compatibility level that your newest version of the Elements system no longer provides a backwards compatible endpoint for.

In each release of the system, Symplectic provides API endpoints for compatibility levels for several earlier versions of the product. This gives you a reasonably long period in which to upgrade your system connections. But don't be caught out.

REFERENCE

Responses

ATOM content

Response content is always either empty, or an ATOM feed. More details about the response feed for each operation is given in the Resources and Operations section.

Without exception, all ATOM-namespaced mark-up should be ignored when consuming the API from a client program.

ATOM content is provided to make it easy for a browser to generate a human readable summary of the information contained in an API response, as a list-based wrapper for detailed API response XML elements that will not appear in browsers or feed readers, and to encourage further development of the API to conform to a simple paginated list-based approach.

Taking any programmatic dependence on the content of ATOM-namespaced elements in API responses is not supported and is highly discouraged. All of the ATOM content is accessible in a more structured manner through embedded API-namespaced content.

API XML schema

The XML schema document provided with this documentation describes all of the API-namespaced request and response XML elements required and returned by the service, all of which belong to the namespace “http://www.symplectic.co.uk/publications/api”.

The schema document contains in depth human readable annotations describing the meaning and use of many of the elements and attributes in the schema.

The schema's target namespace will likely not change as the API evolves. Rather the version number of the schema will change. The version number of the schema appropriate to this documentation is "4.6".

```
<xs:schema version="4.6" xmlns:api="http://www.symplectic.co.uk/publications/api"
xmlns="http://www.symplectic.co.uk/publications/api" elementFormDefault="qualified"
targetNamespace="http://www.symplectic.co.uk/publications/api" xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

The following pages show an example of a full ATOM document response from the API, with the API-namespaced content highlighted in colour and the ATOM-namespaced content greyed out.

GET resource: https://localhost:8091/publications-api/users?&detail=full&per-page=2

ATOM response:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:api="http://www.symplectic.co.uk/publications/api">
```

```
  <api:schema-version>4.6</api:schema-version>
  <category scheme="http://www.symplectic.co.uk/publications/atom/feeds/" term="item-list" label="Item list"/>
  <id>tag:publications@organisation,2009:/publications-api/feeds/objects?categories=users&amp;per-page=2</id>
  <updated>2009-10-29T14:14:12.5471055+00:00</updated>
  <generator uri="https://pubsprd.imperial.ac.uk/" version="3.7">Symplectic Elements</generator>
  <icon>https://localhost:8091/publications-api/symplectic.ico</icon>
  <rights>This feed is the property of the Organisation, and can only be used with permission.</rights>
  <subtitle type="xhtml">
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>
        This feed represents page <a href="https://localhost:8091/publications-
api/objects?categories=users&amp;per-page=2&amp;detail=full">1</a> of 7 listing a total of 13 results from the
following search query: <a href="https://localhost:8091/publications-
api/objects?categories=users&amp;detail=full">https://localhost:8091/publications-
api/objects?categories=users&amp;detail=full</a>
      </p>
    </div>
  </subtitle>
  <api:pagination results-count="13" items-per-page="2">
    <api:page position="this" number="1" href="https://localhost:8091/publications-api/users?per-
page=2&amp;detail=full"/>
    <api:page position="first" number="1" href="https://localhost:8091/publications-api/users?per-
page=2&amp;detail=full"/>
    <api:page position="next" number="2" href="https://localhost:8091/publications-api/users?page=2&amp;per-
page=2&amp;detail=full"/>
    <api:page position="last" number="7" href="https://localhost:8091/publications-api/users?page=7&amp;per-
page=2&amp;detail=full"/>
  </api:pagination>
  <link type="application/atom+xml" rel="self" href="https://localhost:8091/publications-api/ users?per-
page=2&amp;detail=full"/>
  <link type="application/atom+xml" rel="first" href="https://localhost:8091/publications-api/ users?per-
page=2&amp;detail=full"/>
  <link type="application/atom+xml" rel="next" href="https://localhost:8091/publications-
api/users?page=2&amp;per-page=2&amp;detail=full"/>
  <link type="application/atom+xml" rel="last" href="https://localhost:8091/publications-api/
users?page=7&amp;per-page=2&amp;detail=full"/>
  <title>Symplectic Elements search results</title>
  <author>
    <name>Symplectic Elements at Organisation</name>
```

```
  </author>
  <entry>
    <id>tag:publications@organisation,2009:/publications-api/users/5</id>
    <category scheme="http://www.symplectic.co.uk/publications/atom/entries/" term="item" label="Item"/>
    <category scheme="http://www.symplectic.co.uk/publications/categories/" term="user" label="User"/>
    <updated>2009-05-28T14:54:08.937+01:00</updated>
    <link type="application/atom+xml" rel="alternate" href="https://localhost:8091/publications-api/users/5"/>
    <link type="application/atom+xml" rel="related" title="Relationships"
href="https://localhost:8091/publications-api/users/5/relationships"/>
    <title>HOOK, Daniel</title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>User</p>
        <p>
          <a href="https://localhost:8091/publications-api/users/5/relationships">Relationships</a> with other
data
        </p>
      </div>
    </content>
    <api:object category="user" id="5" proprietary-id="3497" authenticating-authority="Internal" username="dwh"
last-modified-when="2009-05-28T14:54:08.937+01:00 href="https://localhost:8091/publications-api/users/5"
created-when="2005-07-28T15:46:18.377+01:00" type-id="1">
      <!--User type 1 is "user"-->
      <api:ever-approved>true</api:ever-approved>
      <api:is-current-staff>true</api:is-current-staff>
      <api:title>Dr</api:title>
      <api:last-name>Hook</api:last-name>
      <api:first-name>Daniel</api:first-name>
      <api:email-address>d.hook@institution.ac</api:email-address>
      <api:organisation-defined-data field-number="1" field-name="CID">Faculty of Natural
Sciences</api:organisation-defined-data>
      <api:organisation-defined-data field-number="2" field-name="Faculty">Department of
Physics</api:organisation-defined-data>
      <api:organisation-defined-data field-number="3" field-name="PrimaryHRO">Theoretical Physics
Group</api:organisation-defined-data>
      <api:organisation-defined-data field-number="4" field-name="LowLevelHRO">South Kensington
Campus</api:organisation-defined-data>
      <api:organisation-defined-data field-number="5" field-name="Campus">Huxley Building</api:organisation-
defined-data>
```

```
      <api:organisation-defined-data field-number="6" field-name="Building">H/508a</api:organisation-defined-data>
      <api:organisation-defined-data field-number="7" field-name="Room">Research Fellow</api:organisation-defined-data>
      <api:organisation-defined-data field-number="8" field-name="PositionName"/>
      <api:organisation-defined-data field-number="9" field-name="Status"/>
      <api:organisation-defined-data field-number="10" field-name="Grading"/>
      <api:relationships href="https://localhost:8091/publications-api/users/5/relationships"/>
    </api:object>
  </entry>
  <entry>
    <id>tag:publications@organisation,2009:/publications-api/users/37674</id>
    <category scheme="http://www.symplectic.co.uk/publications/atom/entries/" term="item" label="Item"/>
    <category scheme="http://www.symplectic.co.uk/publications/categories/" term="user" label="User"/>
    <updated>2009-03-30T16:55:56.577+01:00</updated>
    <link type="application/atom+xml" rel="alternate" href="https://localhost:8091/publications-api/users/37674"/>
    <link type="application/atom+xml" rel="related" title="Relationships" href="https://localhost:8091/publications-api/users/37674/relationships"/>
    <title>KEELE</title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>User</p>
        <p>
          <a href="https://localhost:8091/publications-api/users/37674/relationships">Relationships</a> with other data
        </p>
      </div>
    </content>
    <api:object category="user" id="37674" authenticating-authority="Internal" username="sutherland" last-modified-when="2009-03-30T16:55:56.577+01:00" href="https://localhost:8091/publications-api/users/37674" created-when="2009-02-23T13:16:13.54+00:00" type-id="1">
      <!--User type 1 is "user"-->
      <api:ever-approved>true</api:ever-approved>
      <api:is-current-staff>true</api:is-current-staff>
      <api:last-name>Sutherland</api:last-name>
      <api:email-address>m.sutherland@institution.ac</api:email-address>
      <api:organisation-defined-data field-number="1" field-name="CID"/>
```

```
      <api:organisation-defined-data field-number="2" field-name="Faculty"/>
      <api:organisation-defined-data field-number="3" field-name="PrimaryHRO"/>
      <api:organisation-defined-data field-number="4" field-name="LowLevelHRO"/>
      <api:organisation-defined-data field-number="5" field-name="Campus"/>
      <api:organisation-defined-data field-number="6" field-name="Building"/>
      <api:organisation-defined-data field-number="7" field-name="Room"/>
      <api:organisation-defined-data field-number="8" field-name="PositionName"/>
      <api:organisation-defined-data field-number="9" field-name="Status"/>
      <api:organisation-defined-data field-number="10" field-name="Grading"/>
      <api:relationships href="https://localhost:8091/publications-api/users/37674/relationships"/>
    </api:object>
  </entry>
</feed>
```

The coloured XML in the above example is tightly controlled by the API schema and programmatic clients must restrict their dependence to it.

If any changes are made to the API-namespaced XML, then new documentation will describe the changes. But the greyed-out XML is generated only for browsers and feed readers and is not to be consumed by API clients. Its content can change without notice during patches or upgrades to the system and without being documented.

API content

All API-namespaced content will appear either at the ATOM feed-level or at the ATOM entry-level, with at most one API-namespaced content element inside each ATOM entry.

In this way, the API uses the ATOM format to wrap lists of elements inside ATOM entries, one ATOM entry for each API element in the list.

At the feed level, there will always be a schema-version element that lets you know which version of the API you are using.

This directly corresponds to the version of the schema for the response being made. In the above example ATOM response you will see:

```
<api:schema-version>4.6</api:schema-version>
```

There may also be other feed-level elements.

For example, for resources that represent collections of items, a pagination element describes the size of the results set and your current page within it. As in the above example ATOM response:

```
<api:pagination results-count="578" items-per-page="25">
    <api:page position="this" number="12" href="https://localhost:8091/publications-api/users?page=12"/>
    <api:page position="first" number="1" href="https://localhost:8091/publications-api/users "/>
    <api:page position="previous" number="11" href="https://localhost:8091/publications-api/users?page=11"/>
    <api:page position="next" number="13" href="https://localhost:8091/publications-api/users?page=13"/>
    <api:page position="last" number="24" href="https://localhost:8091/publications-api/users?page=24"/>
</api:pagination>
```

The above example ATOM response wraps a single <api:object> inside each ATOM entry, but other responses will wrap other types of API element.

Error responses

All of the operations are capable of returning an error response, instead of their usual response.

For example, in response to the following GET: https://localhost:8091/publications-api/publications?userame=nonsense

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:api="http://www.symplectic.co.uk/publications/api">
  <api:schema-version>4.6</api:schema-version>
  <category scheme="http://www.symplectic.co.uk/publications/atom/feeds/" term="error-list" label="Error list"/>
  <id>tag:publications@organisation,2009:/publications-api/last-errors</id>
  <updated>2009-10-29T14:28:55.9576745+00:00</updated>
  <generator uri="http://localhost/" version="3.7">Symplectic Elements</generator>
  <icon>https://localhost:8091/publications-api/symplectic.ico</icon>
  <rights>This feed is the property of the Organisation, and can only be used with permission.</rights>
  <subtitle type="xhtml">
    <div xmlns="http://www.w3.org/1999/xhtml">
      <p>This feed describes the errors that occured during the processing of the last request.</p>
    </div>
  </subtitle>
  <title>Errors</title>
  <author>
    <name>Symplectic Elements at Organisation</name>
  </author>
  <entry>
```

```
    <id>tag:publications@organisation,2009:/publications-api/last-errors/1</id>
    <category scheme="http://www.symplectic.co.uk/publications/atom/entries/" term="error" label="Error"/>
    <updated>2009-10-29T14:28:55.9576745+00:00</updated>
    <title>Error: argument fault</title>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>categories value not in the correct format.</p>
      </div>
    </content>
    <api:error code="argument fault">username parameter does not apply to this operation.</api:error>
  </entry>
</feed>
```

The content of an error response is an ATOM feed where the first ATOM entry contains an API error element (and in this case all ATOM entries will contain an error element):

```
<api:error code="argument fault">username parameter does not apply to this operation.</api:error>
```

The possible list of error codes is defined by the XML schema and discussed below. Note that in addition to the error XML returned, you should expect an HTTP error status code to be returned by the server as documented below. In some cases, where a request cannot get through to be executed by API code, you may receive only an HTTP error response.

In particular, if the content of your request contains badly-formed XML (for example, it contains illegal characters such as the vertical tab character at Unicode code point 0x0B) you will receive an immediate HTTP 400 response (Bad Request) before the API has a chance to start executing its application code. If there is a response with this HTTP code but no ATOM error feed then please verify your XML using a suitably configured XML parser.

Error codes

The API error codes are described below. The HTTP status code returned with the response is also listed. Note that the HTTP status code is a less fine-grained indicator of the nature of the error than the application error code, but it can be very useful nonetheless.

Argument fault

HTTP 400 - bad request

One or more of the query string arguments or request body elements is formatted incorrectly, or its value is inappropriate to the operation.

Concurrency conflict

HTTP 409 - conflict

A concurrency conflict (or database deadlock) occurs when two concurrent operations attempt to access data that is being updated by one of them, and the progress of one of the operations is blocked (yours, in this case).

You should simply try the operation again. If this happens often, and you are yourself running concurrent operations, you should either arrange for the concurrent operations to operate on unrelated data, or you should execute your operations serially.

If you are not running concurrent operations yourself, then it is likely that the system administrator has scheduled a heavy data-update operation for the same time as your operations, and it is operating on the same data. You should liaise with the system administrator to adjust the timing of your operations to avoid the conflicts.

Database timeout

HTTP 504 - gateway timeout

A database timeout occurred while attempting to process the request.

This may be due to high server load or simply due to a sudden increase in database activity after a period of idling. Your program may just need to try the operation again once the server has had a few seconds to warm up, or you may need to reduce load by providing the server with a few seconds recovery time at more frequent intervals.

In any case, when writing a client to consume the API, you should build in robustness against database timeouts (by for example implementing a simple exponential back-off strategy) before giving up and reporting an error.

If the timeouts seem to occur at the same time of day each day, they are probably caused by regular server maintenance operations such as database backup plans. You should avoid contact with the system at such times.

If you have a choice, do not force a high server load during office hours as this will heavily impact on the performance of the system at a time when it is needed by the user-interface, which competes for resources with the API.

Remember to liaise with the application administration team to schedule your use of the system.

Invalid credentials

HTTP 401 -unauthorised

The credentials you supplied were invalid or missing, or the IP address of your machine has not been registered with the Elements system. Please contact the system administrator to make sure the credentials you are using are present and correct, and that your IP address is registered. Also check that you are submitting the credentials in the correct way. See the Security section.

If you are sure the credentials are correct, then have the system administrator check the API's log files for a confirmation of why you cannot access the API. The system administrator will need to look up the exact time of your requests in the log, so make sure you make a note of them.

Resource deleted

HTTP 410 - gone

You have requested a resource whose permanent deletion has been logged.

Resource not found

HTTP 404 - not found

You have requested a resource that does not exist in the Elements System.

Server fault

HTTP 500 - internal server error

An error occurred in the server. If retrying the operation does not solve the issue, please contact your Elements System administrator, who will be able to look up the details of the error in the system log. Please inform the system administrator of the exact circumstances of the error, including its time.

Unauthorised access

HTTP 403 - forbidden

The account to which you have authenticated does not have the rights to perform the operation you are attempting. You must contact the system administrator for the Elements System to elevate the permissions associated with your account.

If you find that you are perfectly well able to read data from the API using the various GET-based requests, but that you are unable to execute the POST/DELETE or PUT-based operations to write data to the Element System, then your account needs to be registered at a higher privilege level in the Elements System. Please ask the system administrator to allow your account write-access to the Elements database.

Unlicensed component

HTTP 403 - forbidden

The Elements System supports a modular licensing model. If your institution has not licensed functionality required to service a particular operation request, you will receive this error code.

Success responses

In normal operation, most responses will be success responses, in the HTTP 2xx range, or very occasionally a 303 (see other) redirect, which itself should return in the HTTP 2xx range. A successful response does not preclude the possibility that the operation completed with problems.

Where an operation was completed successfully, but some warnings were generated, such warnings are included in API namespaced elements within each of the ATOM entries.

```
<api:warnings>
  <api:warning associated-field="isbn-13">Failed to parse valid ISBN-13 from ¥"0123456789¥".</api:warning>
</api:warnings>
```

These warnings represent human readable messages suitable for you to log and review by eye should you need to. Where a warning corresponds to a bibliographic record field, the field name is also provided.

Pagination

Resources supporting the page query parameter are implemented as paginated ATOM feeds. All paginated resources offer pagination control and information in the same format and offer navigation of the pages in the same way.

At the feed-level, an API pagination element is returned:

```
<api:pagination results-count="578" items-per-page="25">
  <api:page position="this" number="12" href="https://localhost:8091/publications-api/grants?page=12"/>
  <api:page position="first" number="1" href="https://localhost:8091/publications-api/grants "/>
  <api:page position="previous" number="11" href="https://localhost:8091/publications-api/grants?page=11"/>
  <api:page position="next" number="13" href="https://localhost:8091/publications-api/grants?page=13"/>
  <api:page position="last" number="24" href="https://localhost:8091/publications-api/grants?page=24"/>
</api:pagination>
```

Most of this is self-explanatory. To navigate to the next page, follow the indicated link. The total number of pages, total number of results and the number of items per page are all made available. All elements will always appear for paginated results sets except for the previous and next page elements, which will not appear if there is not a previous or next page respectively.

You can rely on the URI of a particular page of a paginated resource being the same as the URI of the resource itself with a query parameter of "page" included in the query string with the numeric value of the page as its value. So page 3 of /grants is accessed through /grants?page=3. Page numbering begins at 1.

To alter the number of results per page, supply an additional "per-page" query parameter with the required number of items per page as its value. Supplying a value greater than 1000 is an error, and where possible you should keep to a reasonably low a value in order to reduce server load. If using a

full detail response level (i.e. /grants?per-page=25&detail=full), a maximum “per-page” value greater than 25 is an error.

Note that some resources that represent collections are not paginated, and for these you will receive no <api:pagination> element in the response.

Response detail level

Some resources offer control over how much detail is included in the response. The /publications resource by default returns lists of references to publications, and the /{categories}/{id} resources (e.g. /publications/12) in contrast by default returns the full details of the object that the resource represents.

For resources that offer control of detail level, a "detail" query parameter can be used to override the default behaviour of the resource.

The value of the detail parameter should be one of "ref" and "full", according to whether you want abridged reference-style information or full data.

For example, if you are interested in overriding the default behaviour of the /users resource to return lists of users instead of lists of user identifiers then use /users?detail=full.

An example user returned with reference-level detail (detail="ref") looks like this:

```
<api:object category="user" id="5" proprietary-id="3497" authenticating-authority="Internal" username="dwh" last-modified-when="2009-05-28T14:54:08.937+01:00" href="https://localhost:8091/publications-api/users/5" created-when="2005-07-28T15:46:18.377+01:00" type-id="1"/>
```

The same user returned with full-level detail (detail="full") looks like this:

```
<api:object category="user" id="37674" authenticating-authority="Internal" username="sutherland" last-modified-when="2009-03-30T16:55:56.577+01:00" href="https://localhost:8091/publications-api/users/37674" created-when="2009-02-23T13:16:13.54+00:00" type-id="1">
  <!--User type 1 is "user"-->
  <api:ever-approved>true</api:ever-approved>
  <api:is-current-staff>true</api:is-current-staff>
  <api:last-name>Sutherland</api:last-name>
  <api:email-address>m.sutherland@institution.ac</api:email-address>
  <api:organisation-defined-data field-number="1" field-name="CID"/>
  <api:organisation-defined-data field-number="2" field-name="Faculty"/>
  <api:organisation-defined-data field-number="3" field-name="PrimaryHRO"/>
  <api:organisation-defined-data field-number="4" field-name="LowLevelHRO"/>
  <api:organisation-defined-data field-number="5" field-name="Campus"/>
  <api:organisation-defined-data field-number="6" field-name="Building"/>
```

```
  <api:organisation-defined-data field-number="7" field-name="Room"/>
  <api:organisation-defined-data field-number="8" field-name="PositionName"/>
  <api:organisation-defined-data field-number="9" field-name="Status"/>
  <api:organisation-defined-data field-number="10" field-name="Grading"/>
  <api:relationships href="https://localhost:8091/publications-api/users/37674/relationships"/>
</api:object>
```

A response at reference-level detail will only promise to return the top level element and its attributes, whereas a response at full-level detail will return all available data for the object.

Object modification timestamps

API responses include the time of last modification of any object included in the response.

You can use your knowledge of how up-to-date the data you have pulled out of the API is to help you implement differentially updated caches of data by submitting queries to the API to only return data modified since a specified time. This approach can save a lot of bandwidth and I/O resource.

The definition of when an object is considered "modified" is as follows:

- Any of its own values are changed
- Any relationship or relationship suggestion involving the object has any of its values changed
- Any of the objects directly related to the object by a relationship or a relationship suggestion has any of its values changed
- Any new relationship or relationship suggestion involving the object is created
- Any relationship or relationship suggestion involving the object is deleted

The definition of the modification of an object is quite broad, considering an object modified if any of its immediately related neighbours has any of its data values directly changed or if its list of relationships or relationship suggestions changes in any way.

Together with the ability of the API to list objects modified since any particular time of your choosing, this allows you for example to confidently know when to update your cached publication lists. If a user has authored a new publication since you last cached your publication list, you can rely on the author appearing as a modified user in the API the next time you check in looking for changes.

Requests

XML validation

Some operations requiring an XML document in the request content offer a "validate" URL parameter for use during your development phase.

When set to "true", the API will validate your document against the API schema, and will return an argument fault (see the Error Codes section) and a meaningful XML schema validation error message in the response if your document does not validate when compared with the schema.

Making use of this parameter during development can help you to make sure your XML is in line with the expectations of the API while you are testing your code.

This XML validation functionality costs the Elements System significant resources, and must therefore not be used by your code once your client application has gone into production.

Cache control

Not all API resources offer datasets about which the API can confidently know if the content has changed since you last accessed it, but some resources do offer standard HTTP cache control instructions (ETags and Last-Modified headers) to aid you in implementing more bandwidth friendly feeds, such as the individual /{category}/{id} resources.

See the individual operation descriptions for an indication of which of the ETag and Last-Modified caching instructions each operation supports, if any.

Please note that the usual operation success responses can of course be overridden in favour of an HTTP 304 (Not modified) in the case where you choose use HTTP cache control functionality.

API account rights

Note: this section should be ignored if you are connecting to an unsecured endpoint. The API does not perform authentication when accessed through an unsecured endpoint, **and all clients on unsecured endpoints are granted full permissions on all data.**

Each API account comes with a set of rights. You can see the rights assigned to your API account by visiting the /my-account resource.

These rights are:

- whether your account has access to sensitive HR data
- whether your account has access to staff research assessment data
- whether your account is able to modify data in the Elements database

Without the relevant rights, you may receive the standard "unauthorised access" API error response (with HTTP 403 - Forbidden) from some operations, and you may not receive back as much data as you might otherwise expect from other operations. This section describes exactly what behaviour you should expect from each of these rights.

Sensitive HR data

Sensitive HR data is defined as the set of user generic fields numbered 11 upwards. Any data stored in these fields by your institution will not be visible to API accounts without this right.

All operations on the user feed resources /user-feed, user-feed/users/{proprietary-id} and /user-feeds/{feed-id} are inaccessible to API accounts without this right (always a 403 - Forbidden response), and all other operations that return a user object will not include any XML elements representing the user generic fields 11 and higher, even if the values exist in the database.

Research assessment data

Research assessment data is defined as any data represented by the /ra-portfolios, /ra-portfolio and /ra-outputs resources.

All operations on these resources are inaccessible to API accounts without this right (always a 403 - Forbidden response). In addition, the ra-unit-id search parameter on the /users resource cannot be used without this right.

Modifying data

Data modification operations are all those operations whose HTTP method is not "GET" (namely, the "POST", "PUT" and "DELETE" operations). All of these operations will return the standard "unauthorised access" API error response (with HTTP 403 - Forbidden) if accessed from an API account without the right to modify data.

Object identifier format

The Object Identifier Format is supported by many operations and enables you to identify objects in URLs in the way that best suits you. Not all of the operations support the format (e.g. user feed operations, where the user proprietary ID is the only user identifier of use). See the individual operation descriptions.

Example resource URLs supporting this format, with the Object Identifier Format part highlighted in red, are:

/relationships?involving=publication(500) searches for all relationships involving the publication with integer ID 500.

/relationships?involving=user(username-jdjones)&types=8&detail=full searches for all relationships involving the user with username "jdjones".

General format

The general Object Identifier Format must include the category specification, and is

{category}({generalised-id})

For example, user(1) or publication(source-wos,pid-000256633300248).

A valid usage of such an identifier is highlighted in red in the URL relationships?involving=publication(500).

If the category is already specified

Sometimes, the category is already specified in the URL immediately before the placeholder for the identifier. In which case, the Object Identifier Format must exclude the category part (and its brackets), and just use the generalised-id part.

For example, the URL /users/username-jdjones gets the data for the user with username "jdjones". The red-highlighted part of this URL uses this abbreviated Object Identifier Format with the category part not specified.

Generalised ID

Using integer ID

Once the category of the object has been specified, to identify an object by its integer ID, just use the format {id}, where the {id} placeholder is replaced with the numerical ID of the object. The same format can be used for any of the other categories. For example, the following full identifiers (including category) show examples using these integer IDs.

activity(1)
equipment(102)
grant(2034)
org-structure(76)
project(45)
publication(8)
user(450)

Using username

In the context of users, to identify a user by their username instead of by their numerical ID, use the format username-{username} where the {username} placeholder is replaced with the username of the user. Note that sometimes a user cannot be uniquely identified by their username alone, in which case the identifier is treated as not identifying any user.

For example, username-jdjones is a generalised-id for the user with username "jdjones".

Using authenticating authority and username

To identify a user by a combination of their authenticating authority and their username, use the format authority-{authority},username-{username} where the {authority} placeholder is replaced with the authenticating authority of the user, and the {username} placeholder is replaced with the username of the user. Note that the authority and username parts are separated by a comma.

For example, in the context of users, authority-internal,username-jdjones is a generalised-id for the user with authenticating authority "internal" and username "jdjones".

Using a user's proprietary ID

To identify a user by their proprietary ID, use the format pid-{proprietary-id} where the {proprietary-id} placeholder is replaced with the proprietary ID of the user.

For example, in the context of users, pid-0036738 is a generalised-id for the user with proprietary ID "0036738".

Using a record's source-identifier and proprietary ID

To identify an object (except for users) via one of the records it contains, use the format source-{source},pid-{proprietary-id} where the {source} placeholder is replaced with the name of the data source from which the record came, and the {proprietary-id} placeholder is replaced with the proprietary ID of the record (the ID of the record as given to it by the data source). **Note that the source and proprietary ID parts are separated by a comma**.

For example, in the context of publications, source-wos,pid-WOS:000256633300248 is a generalised-id for the publication containing a record from the Web of Science with Thomson UT WOS:000256633300248.

Operations that support the Object Identifier Format allow you to use your favourite way of identifying data when retrieving information about that data.

Resources and operations

The API consists of a set of resources addressable by their URIs, and for each resource a set of operations that can be performed on the resource.

Each operation is described by the URI of the resource on which to operate, the HTTP method (GET, DELETE, PUT or POST) and a request content XML document (in the case of POST or PUT). Please note that all request content XML must be specified in the API XML namespace.

e.g.

- GET /users
- POST <accept/> /suggestions/relationships/8

Where an operation requires request content, such as the POST <accept/> operation above, the definition of the content XML element can be found in the API XML Schema Document supplied with this documentation.

The response to a successful API operation is indicated by the expected HTTP response status code. In this case the expected API-namespaced content of any ATOM feed document in the response body is also returned.

An unsuccessful operation is any operation that does not return one of the expected HTTP success response status codes, and in this case an ATOM error feed is also returned (see the section earlier on Error Responses) if the API began executing application-level code.

Where a URI contains a path element wrapped by braces, it represents a parameter name. Substitute a value directly in to the path of the URI in place of the braced expression. Such a parameter is required and must be URI-encoded.

Where a URI accepts query parameters, they are clearly labelled as either required or optional. Regardless, the order of the parameters does not matter. The parameter values are indicated by curly braces. You should substitute in your desired value in place of the braced placeholders and suitably URI encode your values. For example, if your value includes spaces, you must use the URI space reference encoding of “%20”.

List of resources and operations

Where you see the placeholder {cat} (or {cats}), you must substitute one of the following singular (or plural) category names as appropriate.

{cat}	{cats}
activity	activities
equipment	equipment
grant	grants

org-structure	org-structures
project	projects
publication	publications
user	Users
teaching-activity	teaching-activities

Resource	Verb	Request content	Successful response code	Successful response ATOM entry content	Description
/					Example operations (not documented)
/{cats}	GET		200 (OK)	<object/>	e.g. all publications (/publications) or all users (/users)
/{cats}/{id}	GET		200 (OK)	<object/>	e.g: a publication (/publications/101)
/{cats}/{id}/{cats2}	GET		200 (OK)	<relationship/>	e.g. the publications of a particular user (/users/5/publications)
/{cats}/{id}/relationships	GET		200 (OK)	<relationship/>	e.g. the relationships of a particular user to other data (/users/username-dwh/relationships)
/{cat}/records/{data-source}/{proprietary-id}	GET		200 (OK)	<object/>	e.g. a particular Web of Science record as identified by its Web of Science UT value (/publication/records/wos/WOS:000072950600006)
	PUT	<import-record/>	200 (OK) if updated 201(Created) ifcreated	<object/>	Import a record
	DELETE		204 (No content)		Delete a record
/{cat}/sources	GET		200 (OK)	<data-source/>	e.g. the configured external publication data sources (publication/sources)
/{cats}/{id}/suggestions/relationships/declined	GET		200 (OK)	<relationship-suggestion/>	e.g. All declined relationship suggestions made to a particular user (/users/5/suggestions/relationships/declined)

					e.g.All declined relationship suggestions involving a particular publication (/publications/5672/suggestions/relationships/declined)
/{cats}/{id}/suggestions/relationships/declined/{cats2}	GET		200 (OK)	<relationship-suggestion/>	e.g. All declined publication relationship suggestions made to a particular user (/users/5/suggestions/relationships/declined/publications)
/{cats}/{id}/suggestions/relationships/pending	GET		200 (OK)	<relationship-suggestion/>	e.g. All pending relationship suggestions made to a particular user (/users/5/suggestions/relationships/pending) e.g.All pending relationship suggestions involving a particular publication (/publications/5672/suggestions/relationships/pending)
/{cats}/{id}/suggestions/relationships/pending/{cats2}	GET		200 (OK)	<relationship-suggestion/>	e.g. All pending publication relationship suggestions made to a particular user (/users/5/suggestions/relationships/pending/publications)
/{cat}/types	GET		200 (OK)	<type/>	Field-definitions for the various types of publications (/publication/types). Does not apply to users.
/deleted/{cats}	GET		200 (OK)	<deleted-object/>	e.g. all deleted publications (/deleted/publications)
/deleted/{cats}/{id}	GET		200 (OK)	<deleted-object/>	e.g. a deleted publication (/deleted/publications/56)
/deleted/objects	GET		200 (OK)	<deleted-object/>	All deleted research objects
/groups	GET		200 (OK)	<user-group/>	All the user-groups defined in the system
/herdc/returns	GET		200 (OK)	<herdc-return/>	All HERDC returns in the system

/herdc/returns/{year}	GET		200 (OK)	<herdc-return/>	HERDC return for the specified year
/herdc/returns/{year}/groups	GET		200 (OK)	<herdc-group/>	HERDC groups for the specified year
/herdc/returns/{year}/nominations	GET		200 (OK)	<herdc-nomination/>	All HERDC nominations for the specified year
/herdc/returns/{year}/nominations/{id}	GET		200 (OK)	<herdc-nomination/>	An individual HERDC nomination
/journal/sources	GET		200 (OK)	<journal-source/>	Information about all the sources of authoritative journal data in the system
/journals	GET		200 (OK)	<journal/>	All journals registered in the system
/journals/{issn}	GET		200 (OK)	<journal/>	A journal
/my-account	GET		200 (OK)	<api-account/>	The currently authenticated API account
/objects	GET		200 (OK)	<object/>	All research data objects in the system
/ra-portfolios	GET		200 (OK)	<ra-portfolio/>	All research assessment portfolios in the system
/ra-portfolios/{id}	GET		200 (OK)	<ra-portfolio/>	A research assessment portfolio of a user
/ra-units	GET		200 (OK)	<ra-unit/>	All research assessment units
/relationships	GET		200 (OK)	<relationship/>	All relationships between objects in the system
	POST	<import-user-relationship/>	200(OK) if updated 201(Created) if created	<relationship/>	Import a relationship between a user and a record
/relationships/{id}	GET		200 (OK)	<relationship/>	e.g. A particular relationship between a user and a

					publication (/relationships/44)
	DELETE		204 (No content)		Delete a relationship
	POST	<decline/>	303 (See other) if declined		Converts a relationship to a declined relationship suggestion
	POST	<reoffer/>	303 (See other)		Converts the relationship to a pending relationship suggestion
/relationship-types	GET		200 (OK)	<relationship-type/>	All relationship types
/suggestions/relationships/declined	GET		200 (OK)	<relationship-suggestion/>	All declined suggested relationships between research objects in the system
/suggestions/relationships/pending	GET		200 (OK)	<relationship-suggestion/>	All pending suggested relationships between research objects in the system
/suggestions/relationships/{id}	GET		200 (OK)	<relationship-suggestion/>	A pending or declined relationship suggestion
	DELETE		204 (No content)		Deletes a relationship suggestion
	POST	<accept/>	303 (See other)		Accepts a relationship suggestion, creating a new relationship and deleting the suggestion.
	POST	<decline/>	200 (OK)	<relationship-suggestion/>	Declines a relationship suggestion. The suggestion is transitioned to the declined state.
	POST	<reoffer/>	200 (OK)	<relationship-suggestion/>	Reoffers a declined relationship suggestion. The suggestion is transitioned to the pending state.
/user-feed	GET		200 (OK)	<user-feed-entry/>	The user feed

/user-feed/{partition-id}	GET		200 (OK)	<user-feed-entry/>	A partition of the user feed
	DELETE		204 (No content)		Delete a partition of the user feed
	POST	<import-users-request/>	201 (Created)		Insert new user entries into a partition of the user feed
/user-feed/users/{proprietary-id}	GET		200 (OK)	<user-feed-entry/>	A user in the user feed
	DELETE		204 (No content)		Delete a user from the user feed
	PUT	<user-feed-entry/>	200(OK) if updated 201(Created) if created	<user-feed-entry/>	Import a user to the user feed
/users/{id}/photo	GET		200 (OK)		A user's photo
/users/{id}/search-settings	GET		200 (OK)	<user-search-settings/>	A user's online search settings
	PUT	<user-search-settings/>	200 (OK)	<user-search-settings/>	Update a user's online search settings

Common operation parameters

Many operations offer the same parameters with their usage. Some of these parameters are covered here for convenience.

Parameter: ids

When using this parameter, use of any other parameters except any available detail, page and per-page parameters is an error.

Where available, use this parameter to access a known list of data items with one single query. The format of the ids parameter is a comma-separated list of ids appropriate to the type of data being requested.

For example:

/relationships?ids=1,45,67

The above query would return a feed containing three relationships: the relationships with IDs 1, 45 and 67, in that order.

If any of the data entities do not exist, the results set will not contain them. The order in which the results are returned is the same as the order in which you list the IDs. You can list as many IDs as will not cause your generated URL to be an invalidly large URL.

Please note for a paginated resource, the response is paginated as per normal. If you list more IDs that there are items per page of response, you must paginate through the response to retrieve all of the returned objects.

Parameter: page

This query parameter determines which page of the results set is returned. If omitted, the value "1" is assumed and the API will return the first page of the results.

See the section on Pagination for a discussion of how to navigate through paginated results.

Parameter: per-page

This query parameter determines how many results are returned in each page of the response. If omitted, the value "25" is assumed and the API will return 25 items per page of results.

See the section on Pagination for a discussion of how to navigate through paginated results.

Parameter: detail

This query parameter determines the detail level of the response. If omitted, an operation-specific value of either "ref" of "full" is assumed and the operation will return either references or full details respectively.

See the section on Response Detail Level for a discussion of the detail levels of API responses.

/

The base URI provides an otherwise undocumented front page to the API that can be used to immediately get you up and running by providing you with some links representing sample queries in the system. It is not considered a part of the API and so its response is not documented either here or in the schema document and its content is liable to change at short notice.

/{cats}

This resource represents the objects of a given category, and can be used to search amongst them. For example, /publications represents all publications, and /users all users.

Operation: GET

Use this operation to search for users, publications etc.

Parameters

?ids={idList} Optional: defaults to null
&query={query} Optional: defaults to null
&authority={authority} Optional: defaults to null
&username={username} Optional: defaults to null
&proprietary-id={proprietaryID} Optional: defaults to null
&ra-unit-id={ra-unit-id} Optional: defaults to null
&groups={groups} Optional: defaults to null
&ever-approved={everApproved} Optional: defaults to null
&created-since={createdSince} Optional: defaults to null
&modified-since={modifiedSince} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following URL lists all publications associated with one of your departments:

/publications?groups=12

Parameter: query

The query parameter is used to restrict the response to only those objects matching the query string. If omitted, this restriction is not applied.

The circumstances under which a query string matches an object is a fairly complicated calculation that depends on the category of object being tested and the relationships the object has with others, so that a sensible results set is returned.

/users?query=blood+flow

The above search will return all users who work in the area of "blood flow" (the criterion for which is calculated partly by whether a user is the author of any publications matching the query "blood flow", etc.), whereas the following query will return all publications directly matching the query "blood flow".

/publications?query=blood+flow

Parameter: authority

This parameter can only be used with the /users resource.

This query parameter is used to restrict to users authenticated by the authenticating-authority with the specified ID. If omitted, this restriction is not applied.

The authenticating authority for a user is the system in your institution that is responsible for authenticating the user and is determined by your institution. Use this parameter to distinguish between two users with the same username who “belong” to different authentication systems in your institution. You must contact your Elements system administrator to get the IDs of the relevant authentication authorities in your institution. Note that if the administrator has configured a user to be authenticated by the Elements System itself (i.e. their password is stored in the Elements System and managed by the administrator), then their authenticating authority ID is always the string “Internal”.

Parameter: username

This parameter can only be used with the /users resource.

This query parameter is used to restrict to users to those with the given username. If omitted, this restriction is not applied.

The username is used by the user to log in to the Elements System and is provided by your institution.

Parameter: proprietary-id

This parameter can only be used with the /users resource.

This query parameter is used to restrict to users with the given proprietary ID. If omitted, this restriction is not applied.

The proprietary ID of a user is a unique ID given to the user by your institution. Note that at most one user can be returned if this parameter is supplied.

Parameter: ra-unit-id

This parameter can only be used with the /users resource.

This query parameter is used to restrict to users in a given research assessment unit. If omitted, this restriction is not applied.

The format of the parameter value is a single integer ID value. See the /ra-units resource for the IDs of the research assessment units defined by your institution.

To use this parameter, your API account requires RA rights. See the /my-account resource to view your current rights. Use of the parameter without RA rights will cause an "unauthorised access" error.

Parameter: groups

This query parameter is used to restrict to /users in any one of a collection of user-groups and other /{cats} directly related to any user in the specified collection of groups. If omitted, this restriction is not applied.

The format of the parameter value is a comma-delimited list of group ID values, with no spaces. See the /groups resource for the IDs of the groups defined by your institution.

For example, to return all users who are in either of groups 1 and 5 the following URI should be used:

/users?groups=1,5

To return all publications related to the users in group 12:

/publications?groups=12

Parameter: ever-approved

This query parameter allows you to filter your results according to whether an object has ever been classed as approved research data by your institution. The definition of approved research data is:

- Any user
- Any object imported through the API
- Any object with at least one relationship to another object

There are practical reasons why you might with to restrict to data for which ever-approved=true. For instance, if you are implementing a data synchronisation to another system, you may wish to skip all publications that have never been approved by a user. Such publications are often just false positive search results that have arrived from external data sources.

Parameter: modified since

This query parameter is used to restrict to objects modified since the supplied moment in time. If omitted, this restriction is not applied. Objects created since the specified moment are considered modified, and are included in the results set.

The value of this parameter should be in XML Schema dateTime format, which is inspired by the ISO 8601 standard. Always specify the time-zone. Always specify a time of day down to the second.

Acceptable examples include:

- 2008-06-01T21:36:19.677+01:00
- 2008-03-01T13:00:00Z
- 1999-01-01T00:00:00+00:00

Remember to encode your value for use in a URI. For example:

https://localhost:8091/publications-api/publications?modified-since=2008-03-01T13%3A00%3A00Z

Parameter: created since

This query parameter is used to restrict to objects created since the supplied moment in time. If omitted, this restriction is not applied. See the modified since parameter for details of how to use the parameter.

/{cats}/{id}

This resource represents an individual object in the Elements system. Any valid identifier {id} in the Object Identifier Format can be used.

When using the resource to get an object, the object is described by an `<api:object>` element.

Operation: GET

Use this operation to retrieve a user, publication etc.

Parameters

?detail={detail} Optional: defaults to full

Caching

Supports ETag and Last-Modified headers.

Example

The {id} part of the URL identifies which object is to be viewed using the Object Identifier Format. Examples include:

/users/8
/users/username-jdjones
/users/pid-00412
/publications/1
/publications/source-wos.pid-WOS:000084595500286

Always remember to URL-encode the value of any strings you use.

/{cats}/{id}/{cats2}

These resources represent relationships of an individual object to those in another category. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to list or search amongst all publications related to a particular user, or all projects related to a particular publication, etc.

Parameters

?types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full details of all publication-authorship relationships involving the user with username "jdjones":

GET /users/username-jdjones/publications?types=8&detail=full

See the GET /relationships operation for descriptions of the usage of parameters in common with this operation.

/{cats}/{id}/relationships

These resources represent the relationships of an individual object. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to list or search amongst all objects related to a particular user, or all objects related to a particular publication, etc.

Parameters

?also-involving={object-identifier} Optional: default to null
&types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following URL lists the full object details of all objects related to the user with username "jdjones":

/users/username-jdjones/relationships?detail=full

See the GET /relationships operation for descriptions of the usage of parameters in common with this operation.

/{cat}/records/{source}/{proprietary-id}

This resource represents an individual record within an object in the Elements system, as identified by the proprietary identifier given to it by the data source from which it came.

When using the resource to get a record, the whole object to which the record belongs is in fact returned, and it is returned as an <api:object> element.

Operation: GET

Use this operation to retrieve a record imported from an external data source when you know the record's external identifier.

Parameters

{source} Required: either the string name or integer ID of the external data source can be supplied
{proprietary-id} Required: the identifier of the record as assigned by the external data source
?detail={detail} Optional: defaults to full

Caching

Supports ETag and Last-Modified headers

Example

GET /publication/records/arxiv/abs%2F1203.4219 will return the arXiv record with arXiv identifier abs/1203.4219. Always remember to URL-encode the value of any strings you use as parameters.

Note that you can identify the data source by its ID or by its name. See the API XML schema for more information about data sources.

This operation does not return an <api:record> element as you might expect. Rather it returns an <api:object> element, describing the whole object to which the record belongs. In fact, this resource acts as an alias for the resource representing the object containing the record.

Operation: PUT <import-record/>

Use this operation to create a new record or update an existing record.

Required headers

Content-Type: text/xml

Parameters

?validate={validate} Optional: defaults to false
&detail={detail} Optional: defaults to full

/{cat}/sources

This resource represents the collections of external data sources configured for the specified data category (except for users).

For example, /publication/sources represents the configured publication data sources and will list, amongst others, information about the Web of Science, PubMed and Scopus, as well as the "manual-entry" data source.

Users are not included in this set of resources, since their data model is slightly different from those of the other categories.

Operation: GET

Use this operation to get the details of all data sources for a category of data.

Example

GET /publication/sources

/{cats}/{id}/suggestions/relationships/declined

This resource represents the declined relationship suggestions involving a particular object. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to find all declined relationship suggestions involving a particular object.

Parameters

{id} Required: Object Identifier Format
?types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related object details of all declined relationship suggestions involving user 8:

GET /users/8/suggestions/relationships/declined?detail=full

See the GET /suggestions/relationships/declined operation for descriptions of the usage of parameters in common with this operation.

/{cats}/{id}/suggestions/relationships/declined/{cats2}

This resource represents the declined relationship suggestions involving a particular object, where the other object is restricted to a particular category. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to find all declined publication/activity/etc relationship suggestions involving a particular object.

Parameters

{id} Required: Object Identifier Format
?types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related publication details of all declined publication relationship suggestions involving user 8:

GET /users/8/suggestions/relationships/declined/publications?detail=full

See the GET /suggestions/relationships/declined operation for descriptions of the usage of parameters in common with this operation.

/{cats}/{id}/suggestions/relationships/pending

This resource represents the pending relationship suggestions involving a particular object. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to find all pending relationship suggestions involving a particular object.

Parameters

{id} Required: Object Identifier Format
?types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related object details of all pending relationship suggestions involving user 8:

GET /users/8/suggestions/relationships/pending?detail=full

See the GET /suggestions/relationships/pending operation for descriptions of the usage of parameters in common with this operation.

/{cats}/{id}/suggestions/relationships/pending/{cats2}

This resource represents the pending relationship suggestions involving a particular object, where the other object is restricted to a particular category. Any valid identifier {id} in the Object Identifier Format can be used.

Operation: GET

Use this operation to find all pending publication/activity/etc relationship suggestions involving a particular object.

Parameters

{id} Required: Object Identifier Format
?types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related publication details of all pending publication relationship suggestions involving user 8:

GET /users/8/suggestions/relationships/pending/publications?detail=full

See the GET /suggestions/relationships/pending operation for descriptions of the usage of parameters in common with this operation.

/{cat}/types

This resource above represents the collection of types of object for the given category (except for users).

For example, Users are not included in this set of resources, since their data model is slightly different from those of the other categories.

Operation: GET

Use this operation to get information about all configured object types for a given category.

Example

GET /publication/types will return all configured publication types and will include, amongst others, information about the journal article, software, and conference proceeding types of publications.

/deleted/{cats}

This resource represents all deleted objects of the given category.

Operation: GET

Use this operation to find out which objects of a given category have been deleted. You might use the results of this operation to reflect deletions in a system that caches Elements research data.

Parameters

?deleted-since={deletedSince} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25

Example

The following operation lists all publications deleted since a given moment in time:

GET /deleted/publications?deleted-since=2009-10-25T13%3A34%3A00Z

/deleted/{cats}/{id}

This resource represents a deleted object.

Operation: GET

Use this operation to find out information about a deleted object, such as when it was deleted.

Example

The following operation lists the details of the deletion of publication 85:

GET /deleted/publications/85

See the schema for more information about the data returned for deleted objects.

/groups

This resource represents all the administrative user groups defined within Elements.

Operation: GET

Use this operation to list all the user groups defined in the system.

/herdc/returns

This resource represents all of the HERDC returns defined in the system.

Operation: GET

Use this operation to find all HERDC returns in the system.

/herdc/returns/{year}

This resource represents a HERDC return defined in the system. Any valid return {year} in the format yyyy can be used. The resource does not exist if no return for the specified year has been created.

Operation: GET

Use this operation to find a single HERDC return.

Example

The following operation gets the HERDC return for 2012:

GET /herdc/returns/2012

/herdc/returns/{year}/groups

This resource represents the HERDC groups defined for a given HERDC return. Any valid return {year} can be used.

Operation: GET

Use this operation to find all groups for a HERDC return.

Example

The following operation gets the HERDC groups for return year 2012:

GET /herdc/returns/2012/groups

/herdc/returns/{year}/nominations

This resource represents the HERDC nominations for a return defined in the system. Any valid return {year} can be used.

Operation: GET

Use this operation to find all nominations for a HERDC return.

Parameters

?status={status} Optional: defaults to null
&category={category} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation gets the HERDC nominations in full detail for return year 2012 with status accepted and category 157:

GET /herdc/returns/2012/nominations?status=accepted&category=157?detail=full

The following operation returns all nominations for the 2012 HERDC return:

GET /herdc/returns/2012/nominations

Parameter: status

Use this parameter to restrict the returned nominations to those whose status matches the supplied value. See the API schema document for possible values of nomination status.

For example:

/herdc/returns/2012/nominations?status=declined

The above query would return a feed containing all declined HERDC nominations for the 2012 return.

Parameter: category

Use this parameter to restrict the returned nominations to those of a given HERDC category. See the parent /herdc/returns/{year} resource for details of the configured categories for the relevant return. Such categories are identified numerically.

For example:

/herdc/returns/2012/nominations?category=157

The above query would return a feed containing all “category 157” HERDC nominations for the 2012 return. Your administrator will previously have defined category 157 of the HERDC 2012 return to mean, perhaps, “journals”.

/herdc/returns/{year}/nominations/{id}

This resource represents a single HERDC nomination defined in the system. Use any valid return {year} and nomination {id}.

Operation: GET

Use this operation to view a single HERDC nomination.

Example

The following operation returns the nomination with ID 29 belonging to the 2012 HERDC return:

GET /herdc/returns/2012/nominations/29

/journal/sources

This resource represents the collection of external data sources that have provided Symplectic Elements with authoritative journal data.

You can use the data provided by these journal information sources wherever you see an ISSN value in bibliographic data. Use the GET /journals/{issn} operation to see all data available from all of the journal data sources used by Symplectic Elements for a journal.

GET /journal/sources will list, amongst others, information about the Thomson Reuters JCR dataset, the Science-Metrix journal dataset, and several others.

Operation: GET

Use this operation to get the details of all journal information sources.

Example

GET /journal/sources

/journals

This resource represents all journals registered in the system from external journal data sources whose data is included in the system. See the /journal/sources resource for a complete list of the journal data sources whose data is included within Symplectic Elements.

Operation: GET

Use this operation to list or search for journals.

Parameters

?issns={issns} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25

Example

The following operation lists all journals matching any of the provided ISSNs:

GET /journals

Parameter: issns

Use this parameter to access a known list of journals with one single query. The format of the ids parameter is a comma-separated list of journal ISSNs. Each ISSN **must be a valid ISSN**. For example, 1234-5679 is a valid ISSN. 1234-5678 is an invalid ISSN. See http://en.wikipedia.org/wiki/International_Standard_Serial_Number.

For example:

/journals?issns=1234-5679,1954-5568

The above query would return a feed containing up to two journals, depending on whether any of the journal data sources contains data for the specified journals..

/journals/{issn}

This resource represents information about a single journal as provided by all of the journal data sources used by Symplectic Elements. The journal is identified by ISSN.

Wherever the bibliographic data of a publication record contains an ISSN field whose value corresponds to a journal for which there is journal information provided by at least one the registered journal data sources, a convenient hyperlink is provided with the ISSN value that dereferences to this resource. In this way, you can quickly discover all available authoritative information about a publication's parent journal. See the XML Schema section describing bibliographic records for more information about where to find this link

Operation: GET

Use this operation to retrieve information about a journal.

Example

GET /journals/1234-5679

/my-account

This resource represents the details of the currently authenticated API account.

Operation: GET

Use this operation to see the details of the currently authenticated API account. The response includes your account username, and details of

- whether your account has access to sensitive HR data
- whether your account has access to staff research assessment data
- whether your account is able to modify data in the Elements database

See the section on API account rights for more information about what these rights represent.

Note: if you are accessing the API via an unsecured endpoint, this page will display the details of an anonymous account, which is shown to have been granted none of the available rights. However, as will be noted on the page itself, all account-based rights checks are bypassed on an unsecured endpoint.

/ra-portfolios

This resource represents the entire collection of research assessment portfolios in the Elements system, and can be used to search the portfolios.

Operation: GET

Use this operation to retrieve the list of all research assessment portfolios.

Parameters

?ids={idList} Optional: defaults to null
&modified-since={modifiedSince} Optional: defaults to null
&involving-user={object-identifier} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists all research assessment portfolios associated with the user with username "sjones":

GET /ra-portfolios?involving-user=username-sjones

Parameter: ids

Use this parameter to access a known list of portfolios with one single query. The format of the ids parameter is a comma-separated list of integer IDs.

Parameter: modified since

This query parameter is used to restrict to portfolios modified since the supplied moment in time. If omitted, this restriction is not applied. Portfolios created since the specified moment are considered modified, and are included in the results set.

For further details, see the documentation for the parameter of the same name for the /objects resource.

Parameter: involving-user

Restricts the portfolios returned to those of the specified user. Currently, only at most one portfolio can exist per user. To specify the user, use the Object Identifier Format.

/ra-portfolios/{id}

This resource represents an individual research assessment portfolio belonging to a user.

Operation: GET

Use this operation to retrieve a single research assessment portfolio. To find a portfolio belonging to a particular user, use the GET /ra-portfolios resource.

Parameters

?detail={detail} Optional: defaults to full

Caching

Supports ETag and Last-Modified headers.

/ra-units

This resource represents the collection of research assessment units that your institution has configured in the system.

Operation: GET

Use this operation to list the research assessment units defined in your institution. You can use the IDs of the returned RA units to search for users belonging to the unit. See the GET /users operation for more details.

Caching

Supports ETag and Last-Modified headers.

/relationships

This resource represents the entire collection of relationships between objects in the system, and can be used to search for and update relationship data. One example is to search for all publications authored by a particular user.

When searching for relationships you may find it more convenient to use the GET /{cats}/{id}/relationships or GET /{cats}/{id}/{cats2} resources.

Operation: GET

Use this operation to list all relationships in the system, or search for relationships involving a given object or between a given pair of objects. You can also restrict by relationship type (such as publication authorship).

Parameters

?ids={idList} Optional: defaults to null
&involving={object-identifier} Optional: defaults to null
&also-involving={object-identifier} Optional: defaults to null
&types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full publication details of all publications authored by the user with username "jdjones":

GET /relationships?involving=user(username-jdjones)&types=8&detail=full

Note that this search can also be achieved with the operation GET /users/username-jdjones/publications?types=8.

Parameter: involving

Restricts the relationships returned to those involving the specified object. To specify the object, use the Object Identifier Format.

Parameter: also-involving

If using this parameter, you must also use the involving parameter.

Further restricts the relationships returned to those also involving the specified second object. To specify the second object, use the Object Identifier Format. If you specify the same object as is specified with the "involving" parameter, only self-referencing relationships are returned.

Parameter: types

The types parameter can be used to restrict the response to relationships that are of any of the listed types. If omitted, this restriction is not applied.

The format of the types parameter is a comma-delimited list of relationship type IDs (integers) with all whitespace trimmed.

For example, an operation to return all publications authored or edited by the user with ID 5 is:

/relationships?involving=user(5)&types=8,9

The available relationship types and their IDs can be looked-up using the GET /relationship-types operation.

Parameter: detail

This query parameter determines the detail level of the objects returned inside the relationship elements.

Operation: POST <import-user-relationship>

Use this operation to import (create or update as appropriate) a relationship between a given user and the parent object of a given record.

This operation provides you with a very similar ability to import data as is automatically achieved through synchronisation with the various online bibliographic data sources. See the Examples section for an overview of how to implement a complete data feed into the system, including relationship imports.

The operation is idempotent, and can therefore be used to either create new or update previously imported relationships. Use this operation for example to inform the Elements System of which previously imported publications are authored by which users of the system, or which users are the primary investigators of previously imported grants.

See the Examples section for a walk-through of the request data you must supply when using this operation.

Required headers

Content-Type: text/xml

Parameters

?validate={validate} Optional: defaults to false
&detail={detail} Optional: defaults to full

Successful responses

HTTP 201 (Created) is returned if a new relationship was created, and the "Location" header is set to the URL of the created relationship.

HTTP 200 (OK) is returned if an existing relationship was updated as a result of the operation, and a representation of the relationship is returned in the response content.

/relationships/{id}

This resource represents a single relationship between two objects in the system (such as the authorship of a publication by a user).

Operation: GET

Use this operation to retrieve the details of a relationship between to objects, and optionally full details of the objects themselves.

Parameters

?detail={detail} Optional: defaults to full

Parameter: detail

At full detail level, the relationship will contain the full details of both objects involved in the relationship.

Operation: DELETE

Use this operation to delete a relationship from the system. Once a relationship is deleted, it is gone forever, so use this operation with extreme care.

It is often more appropriate to decline a relationship than it is to delete it. To decline a relationship, see the POST <decline/> /relationships/{id} operation. Declining a relationship has the advantage of letting the system remember that the relationship should not be reoffered to users of the system as a suggestion.

You could use this operation to correct previously uploaded relationships that were uploaded in error, though you should treat any deletion of data as a last resort, take appropriate steps to back up your database before deleting any data, and conduct a thorough review of data integrity after running any deletions. Always work closely with your system administrator when deleting data.

It is almost always better to manually manage problem data items than to write programs to delete them via this API.

A successful response will be returned whether or not the identified relationship existed at the time the operation was executed, in line with standard REST practice.

Operation: POST <decline/>

Use this operation to decline a relationship.

If the relationship type is supported by the Symplectic Elements suggestion framework, the relationship is atomically deleted and replaced by a declined relationship suggestion. Otherwise an argument fault error response is returned.

See the results of the GET /relationship-types operation for information about which relationship types are supported by the Elements suggestion framework. Such relationships include publication authorship.

The system will remember the declined suggestion and will not automatically reoffer the relationship to users of the system as a suggestion.

Required headers

Content-Type: text/xml

Successful responses

HTTP 303 (See other) is returned if the relationship was deleted and a new declined suggestion was created, and the "Location" header is set to the URL of the newly created declined relationship suggestion.

HTTP 204(No content) is returned if the relationship was simply deleted.

Operation: POST <reoffer/>

Use this operation to reoffer a relationship as a pending relationship suggestion.

If the relationship is of a relationship type that is supported by the Symplectic Elements suggestion framework, the relationship is atomically deleted and replaced by a pending relationship suggestion. Otherwise an argument fault error response is returned.

See the results of the GET /relationship-types operation for information about which relationship types are supported by the Elements suggestion framework. Such relationships include publication authorship.

Required headers

Content-Type: text/xml

Successful responses

HTTP 303 (See other) is returned if the relationship was deleted and a new pending suggestion was created, and the "Location" header is set to the URL of the newly created pending relationship suggestion.

/relationship-types

This resource makes available the full details of all relationship types currently defined in the system.

The id value of a relationship types you are interested in can then be used by you as a search criterium when for example searching the database using the GET /relationships operation.

For example, your client system may represent the web pages of your institution. You may wish to cache the list of publications authored by the staff in your institution, for which you would need to know which relationship represents "publication authorship". Use this resource to look up which relationship type this is before your program your client.

Operation: GET

Use this operation to return the list of all relationship types in the system.

/suggestions/relationships/declined

This resource represents all declined relationship suggestions.

Operation: GET

Use this operation to list or search amongst all declined relationship suggestions.

Parameters

?ids={idList} Optional: defaults to null
&types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related object details of all declined publication authorship suggestions in the system:

GET /users/8/suggestions/relationships/declined?types=8&detail=full

/suggestions/relationships/pending

This resource represents all pending relationship suggestions.

Operation: GET

Use this operation to list or search amongst all pending relationship suggestions.

Parameters

?ids={idList} Optional: defaults to null
&types={types} Optional: defaults to null
&page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25
&detail={detail} Optional: defaults to ref

Example

The following operation lists the full relationship suggestion and related object details of all pending publication authorship suggestions in the system:

GET /users/8/suggestions/relationships/pending?types=8&detail=full

/suggestions/relationships/{id}

This resource represents a single pending or declined relationship suggestion between two objects in the system (such as the authorship of a publication by a user).

Operation: GET

Use this operation to retrieve the details of a relationship suggestion between to objects, and optionally full details of the objects themselves.

Parameters

?detail={detail} defaults to full

Parameter: detail

At full detail level, the relationship suggestion will contain the full details of both objects involved in the relationship.

Operation: DELETE

Use this operation to delete a pending or declined relationship suggestion from the system.

It is often more appropriate to decline a pending relationship suggestion than it is to delete it. To decline a relationship suggestion, see the POST <decline/> /suggestions/relationships/{id} operation. Declining a pending relationship suggestion has the advantage of letting the system remember that the relationship should not be reoffered to users of the system.

A successful response will be returned whether or not the identified relationship suggestion existed at the time the operation was executed, in line with standard REST practice.

Operation: POST <accept/>

Use this operation to accept a pending or declined relationship suggestion. The relationship suggestion is deleted and replaced by an actual relationship. The HTTP 303 (See other) redirect response points to the resource representing the newly created relationship.

Required headers

Content-Type: text/xml

Operation: POST <decline/>

Use this operation to decline a pending relationship suggestion.

The pending relationship suggestion is transitioned to a declined relationship suggestion. The system will remember the declined suggestion and will not automatically reoffer the relationship to users of the system as a pending suggestion.

Required headers

Content-Type: text/xml

Operation: POST <reoffer/>

Use this operation to reoffer a declined relationship suggestion.

The declined relationship suggestion is transitioned to a pending relationship suggestion.

Required headers

Content-Type: text/xml

/user-feed

This resource represents the collection of all entries in the system user feed table. See the HR Feed Guide for more information about the user feed functionality in the Elements System.

Operation: GET

Use this operation to see all of the entries in the user feed table. See the section on "Implementing a user feed" for more information.

Parameters

?page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25

/user-feed/{partition-id}

Some institutions prefer to treat the user feed as if it were split into separate partitions, each of which is updated in bulk by a separate client feed system and accessed by its partition ID (or feed-id). This resource represents such a partition of the user feed and its bulk update operations.

Use this resource to view the contents of a user feed partition, to clear a partition, or to import new entries into a partition.

A logically infinite number of user feed partitions "exists", and so any value for {partition-id} can be supplied. For a value not previously seen, the user feed partition is considered to be empty rather than non-existent.

Operation: GET

Use this operation to view the user feed entries in a user feed partition. See the section on "Implementing a user feed" for more information.

Operation: DELETE

Use this operation to delete all user feed entries within a user feed partition. See the section on Implementing a user feed for more information.

Operation: POST <import-users-request>

Use this operation in combination with DELETE /user-feed/{partition-id} to implement a bulk partition based user feed to the Elements system.

The operation allows you to feed the Elements System in bulk with new user entries. See the section on Implementing a user feed for more information.

Required headers

Content-Type: text/xml

Parameters

?validate={validate} Optional: defaults to false

/user-feed/users/{proprietary-id}

This resource represents the entries in the user feed for an individual user. Use this resource's operations to see the status of a user in the user feed, or to add, update or remove users from the user feed one at a time

Operation: GET

Use this operation to see the status of a user in the user feed, or to add, update or remove users from the user feed one at a time.

This operation will return one entry for each user feed entry in the user feed for the indicated user.

Although it is not ever desirable for there to be more than one user feed entry for any particular user in the user feed, it is possible for you to end up achieving this if you have not configured your user feed data properly. For this reason, this operation can return more than one user-feed-entry element.

Parameters

?page={page} Optional: defaults to 1
&per-page={perPage} Optional: defaults to 25

Operation: DELETE

Use this operation to delete all user feed entries for the indicated user. Even if no user feed entries exist for the user, a successful response is returned.

Operation: PUT <user-feed-entry>

Use this operation to import (create or update) a user feed entry in the user feed.

If no user feed entries yet exist for the user, this operation will add the supplied user feed entry to the user feed.

If one or more user feed entries already exist for the user, the user feed is updated to contain just the supplied single user feed entry for this user, effectively deleting all existing entries for this user and replacing them with the new entry.

Required headers

Content-Type: text/xml

Parameters

?validate={validate} Optional: defaults to false

Successful responses

HTTP 201 (Created) if the user did not previously exist in the user feed, and a "Location" header set to the URL of the created user feed entry.

HTTP 200 (OK) if the user was already in the user feed, and has had their entry (entries) replaced by the provided data.

/users/{id}/search-settings

This resource represents the online search settings of the user identifier by the given ID, which supports user IDs in the Object Identifier Format.

Operation: GET

Use this operation to view the current search settings for a user.

Example

The following operation gets the search settings for the user with ID 512:

GET /users/512/search-settings

Operation: PUT <user-search-settings>

Use this operation to modify the search settings for a user.

To use this operation, the supported workflow is:

1. Retrieve the <user-search-settings> XML element returned by the corresponding call to GET /users/{id}/search-settings.
2. Modify the contents of the element in situ in your client program to your new desired settings, following advice included in the XML schema for the <user-search-settings> element.
3. Call this operation with your modified <user-search-settings> XML element as the supplied content.
4. Check the response code for any “argument fault” errors returned to your by the operation, and log any error messages.
5. Review your logs regularly.

As with any development against operations that modify data in Elements by supplying XML content, use the “validate” query parameter until you are confident that your XML does not break any constraints required by the API’s XML schema. When deploying your code in production, you must disable use of the “validate” query parameter.

User search settings are quite complicated, necessitating the workflow above, which limits your chance of submitting invalid settings for the various data sources by presenting you in step 1 with a valid starting point for the document that you supply in step 3. **You may only modify the parts of the document supplied in step 1 that are documented as “Editable” in the API XML schema**.

Note that upon successful modification of a user’s search settings, the user will be placed into the Elements search queue. Some time thereafter, the system will perform a search using the new search settings.

Required headers

Content-Type: text/xml

Parameters

?validate={validate} Optional: defaults to false

Successful responses

HTTP 200 (OK).

/users/{id}/photo

This resource represents the photo of the user with the given ID and returns a photo file in jpg, gif, or png format. Not all users may have a photo. For users without a photo, a standard “Resource not found” error response is made (including an HTTP 404 “Not found” status code).

Operation: GET

Use this operation to retrieve the photo of a user.

Example

The following operation gets the user photo for the user with ID 512:

GET /users/512/photo

Overview of the Elements Database

Objects and categories

The Elements database manages seven broad categories of research object and the relationships between them.

The seven categories of objects are: professional activities, pieces of equipment, grants, organisational structures, projects, publications, and users.

A large number of possible relationships can exist between the objects in the system, one example of which is the authorship of a publication by a user.

Activities

Each professional activity is one of a number of types of activity defined by your institution.

Example types are: award, commercial spinoff, editorial board, external collaboration, external committee, fellowship, guest lecture, industrial connection, link with another academic body, membership of a professional body.

Equipment

A piece of equipment might share relationships with projects and members of staff (users), or might be funded by grants.

Grants

Grants are awards of or applications for funding.

Organisational structures

Each organisational structure object might represent a department or faculty in your institution, or a department or faculty within an external body, or might represent an external body itself.

Projects

A project object, like any other object, might share any number of relationships with other objects, some of which might represent grants that fund the project, or managers of the project.

Publications

Each publication is one of a number of types of publication defined by your institution.

Example types are: artefact, book, book chapter, composition, conference proceeding, design, exhibition, internet publication, journal article, other, patent, performance, poster, report, scholarly edition, software, thesis/dissertation.

Teaching Activities

Each teaching activity is one of a number of types of teaching activity defined by your institution.

Users

Each user object represents a user of the system, past or present.

Your institution controls the list of users in the in a number of ways: manually, using the administrative tools available in the web application, and automatically, by providing user feeds through this API.

You may be reading this document because you are to provide a user feed to the system through the API.

Records

Because research data can enter the system from more than one data source, each object (except for users) has the ability to store as many records (each representing the very same object) as there are data sources from which the object has been discovered/imported. This is a central feature of the Elements system, and is one the foundations upon which its disambiguation functionality works.

Types

Each category (except users) distinguishes between one or more types of object within the category. For example, a publication object could be of type "book" or "journal article".

The definition of a type includes a complete specification of the fields stored in the database for each instance of the type, and your institution has a degree of control over the set of types for each category and the settings for each type definition.

The API offers resources for you to interrogate the definition of these types so that you can be aware of the fields stored for each type of object within each category. For example, the /publication/types resource represents the publication types configured in the system, and lists the data fields used by those types.

Your institution might decide to alter the definition of any of the types in the system, perhaps adding a new field called "Comments" to the patent type of publication, for example. All patents stored in the Elements database will then have a "Comments" field whose value will be immediately available to you through the API.

The following extract from an API response shows some of the fields of the "artefact" type of publication.

```
<api:type id="12">
  <api:heading-singular>Artefact</api:heading-singular>
  <api:heading-plural>Artefacts</api:heading-plural>
  <api:heading-lowercase-singular>artefact</api:heading-lowercase-singular>
  <api:heading-lowercase-plural>artefacts</api:heading-lowercase-plural>
  <api:fields>
    <api:field>
      <api:name>title</api:name>
      <api:display-name>Title</api:display-name>
      <api:type>text</api:type>
      <api:is-mandatory>true</api:is-mandatory>
    </api:field>
    <api:field>
      <api:name>abstract</api:name>
      <api:display-name>Abstract</api:display-name>
      <api:type>text</api:type>
      <api:is-mandatory>false</api:is-mandatory>
    </api:field>
    <api:field>
      <api:name>authors</api:name>
      <api:display-name>Authors</api:display-name>
      <api:type>person-list</api:type>
      <api:is-mandatory>true</api:is-mandatory>
    </api:field>
    <!--more fields-->
  </api:fields>
</api:type>
```

Each object type has an integer ID that is unique within the category to which it belongs.

If your institution makes changes to the type definitions in the Elements System, you may need to update any client programs that you have written that consume API data. Make sure you and your system administrator are in dialogue about any type changes planned by your institution.

Data sources

As mentioned, each object (except for a user) is capable of containing as many records representing the same object as there are registered sources from which the system gets its data. Each record independently represents the object's metadata as imported from the respective data source. The Elements system always knows where each record came from.

For example, your institution might have the following publication data sources configured, and each publication will then contain zero or more records sourced from each of them: arXiv, DBLP, manually-entered, PubMed, Scopus, Mendeley and the Web of Science.

The Elements System does allow for multiple records from the same data source to represent the same object, but only for some sources.

When you consume the data for a publication through the API, you are able to choose which record best suits your needs. If you prefer the data from PubMed, you are free to use it if it is present. You might prefer to use all of the records in some way. You might choose the record declared as the "preferred record" of one of the publication's authors.

The choice is yours. If you prefer not to deal with the hassle of multiple records, you can always just read the first one.

All objects will have at least one record, except for users (whose data is represented slightly differently to those of all other categories).

Here is an example publication with multiple records. Note that for the sake of brevity we have stripped out all of the fields except for authors and titles.

The "authors" and "title" fields are defined by the type 5 publication (the "journal article" type). The type of the publication is indicated by the "type-id" attribute on the api:object element.

```
<api:object category="publication" id="15" last-modified-when="2009-05-28T14:54:08.937+01:00" href="https://localhost:8091/publications-api/publications/15" created-when="2006-07-05T00:20:20.833+01:00" type-id="5">
  <!--Publication type 5 is "journal article"-->
  <api:ever-approved>true</api:ever-approved>
  <api:reporting-date-2>1986-04-11</api:reporting-date-2>
  <api:records>
    <api:record format="native" source-id="3" source-name="wos" source-display-name="Web of Science" id-at-source="A1986C172200006">
      <api:citation-count>25</api:citation-count>
      <api:verification-status>verified</api:verification-status>
      <api:native>
```

```
        <api:field name="authors" type="person-list" display-name="Authors">
          <api:people>
            <api:person>
              <api:last-name>GRESTY</api:last-name>
              <api:initials>M</api:initials>
            </api:person>
            <api:person>
              <api:last-name>BRONSTEIN</api:last-name>
              <api:initials>A</api:initials>
            </api:person>
          </api:people>
        </api:field>
        <api:field name="title" type="text" display-name="Title">
          <api:text>OTOLITH STIMULATION EVOKES COMPENSATORY REFLEX EYE-MOVEMENTS OF HIGH-VELOCITY WHEN LINEAR MOTION OF THE HEAD IS COMBINED WITH CONCURRENT ANGULAR MOTION</api:text>
        </api:field>
        <!--other fields-->
      </api:native>
    </api:record>
    <api:record format="native" source-id="2" source-name="pubmed" source-display-name="PubMed" id-at-source="3714102">
      <api:verification-status>verified</api:verification-status>
      <api:native>
        <api:field name="authors" type="person-list" display-name="Authors">
          <api:people>
            <api:person>
              <api:last-name>Gresty</api:last-name>
              <api:initials>M</api:initials>
            </api:person>
            <api:person>
              <api:last-name>Bronstein</api:last-name>
              <api:initials>A</api:initials>
            </api:person>
          </api:people>
        </api:field>
        <api:field name="title" type="text" display-name="Title">
          <api:text>Otolith stimulation evokes compensatory reflex eye movements of high velocity when linear
```

```
motion of the head is combined with concurrent angular motion.</api:text>
        </api:field>
        <!--other fields-->
      </api:native>
    </api:record>
  </api:records>
  <api:relationships href="https://localhost:8091/publications-api/publications/15/relationships"/>
</api:object>
```

Initial grant data sources

The three grant data sources configured with a default installation of the Elements System are: Symplectic Grants Online, manually-entered, and Institutional Grants System.

Symplectic Grants Online aggregates grant data from the major UK research funding councils and provides an interface to this system for the automated detection of grants relevant to your users.

The Institutional Grants System data source is a placeholder to represent your existing grants system (if any). You should for example use the identifier for this data source when importing data from your existing grants system into the Elements system. This way, the Elements system knows where the data came from.

See the API schema for more details about data sources.

Initial publication data sources

Example data sources configured with a default installation of the Elements System are: arXiv, Cinii EN, Cinii JP, DBLP, manually-entered, figshare, Mendeley, PubMed, RePEc, SciVal Experts, Scopus, the Web of Science and the Web of Science (Lite).

```
<api:record format="native" source-id="3" source-name="wos" source-display-name="Web of Science" id-at-source="A1986C172200006">...
...
<api:record format="native" source-id="2" source-name="pubmed" source-display-name="PubMed" id-at-source="3714102">...
```

From the above publication XML example's record/@source-name attributes we can see that the two records are from the Web of Science, and PubMed, respectively.

When consuming this publication, you have the choice to use either record, or a mixture of them both.

See the schema documentation for more details about data sources.

Initial data sources for other categories

All categories (except users) ship with two data sources configured: the manually-entered data source, which is always assigned a name of "manual" and the Institutional Source.

The manually-entered data sources (one for each category, except users) typically represent data entered into the system manually by the users of the system at your institution (perhaps by file-upload, or by directly typing their details into the user interface). Often, an institution will also wish to import data from legacy systems into the Elements system flagged as manually-entered records.

The Institutional Source data source is a placeholder to represent your existing data system (if any). You should for example use the identifier for this data source when importing data from your existing system into the Elements system. This way, the Elements system knows where the data came from.

Unless the data was directly transferred from an online data source such as arXiv, or a dedicated automated feed from systems at your institution, data uploaded into the system by users, or in a one-off manner in bulk by your institution, is usually considered manually-entered.

More information about data sources

More information about the data sources currently configured at your institution is made available through the API itself. See the /data-sources resource.

Deduplication

From time to time, the system will itself notice, or be told by an academic user or librarian, that two existing objects in the system refer to the same thing. For example, two publication objects might refer to the same published item in the real world.

The system may prompt interested users to merge the two objects in to one. This process is crucial to maintaining the accuracy of the data in the system.

Although it may not affect your particular use case as a consumer of the API, it may be helpful to have a brief overview of what happens during this process.

The newer of the objects is selected to be the “source”, and the older object is selected to be the “destination” of the merge. All of the records of the source object are then transferred into the destination object, and the source object is deleted.

Any existing relationships of the source and destination objects to other objects in the system are managed during the merge process to most effectively preserve meaningful data. From the point of view of an API consumer, the “source” object will have been deleted, the “destination” object will have been modified, all relationships involving the "source" object will have been deleted, and the collection of relationships to the "destination" object may have been modified.

Occasionally, the reverse (a “split”) will occur.

A user of the system may notice that one record belonging to an object does not belong with the others, since it obviously refers to a different external entity than the others.

In this case, the offending record is removed from the object and placed inside a new object of its own. From the point of view of a consumer of the API, the original object will have been modified, and a new object will have been created.

For some sources, objects cannot contain more than one record, therefore it is not always possible to merge two objects that both contain a record from that data source.

Deleted objects

For various reasons in the normal course of events, objects will be deleted from the system.

Because some external systems need to be able to know when an object has been deleted, information about the object will be accessible through the API in resources under the /deleted/{cats} part of the URI space, though the original resource representing the object will return a 410 (Gone) HTTP response.

Whenever a deleted object is requested from a URL under the /deleted/{cats} part of the URI space, the level of detail returned will be just enough to determine the identity of the object, plus a little extra administrative information. No nested elements will be provided, since the item has actually been deleted and the data is no longer available.

```
<api:deleted-object category="publication" id="10" deleted-when="2009-02-10T19:57:11+00:00" href="https://localhost:8091/publications-api/deleted/publications/10"/>
```

Identifying objects

Each object has an ID given to it by the Element System. This is referred to simply as the object's ID and it is always an integer.

The value is persistent and unique only within the object's category, never changing or being reused by another object within the same category.

If the object is ever deleted, the API will still be able to tell you that the object has been deleted if you ask for the object of the same category with the same ID.

The category/ID pair is persistant and unique within the entire system and so the category/ID pair is the true unique identifier of the object.

```
<api:object category="publication" id="10"...
```

Users can also be assigned a Proprietary ID by your institution (an institution-wide staff-id or human-resources-id). For users with a Proprietary ID, it is a string value unique amongst all users that have ever been assigned one, but users are not required to have a Proprietary ID.

The Proprietary ID is passed to the Elements System by your institution when automatically feeding users in to the system using the API. By storing the Proprietary ID provided by your user feeds, the Elements System knows which users to update when their details change in the user feed you provide.

Each user is additionally assigned a username and authenticating authority pair, also given to him/her by your institution.

The username is unique amongst all active users that share the same authenticating authority and is used by the user to log in to the Elements System user interface (an active user is one who is not deleted and who is either a current member of staff or has not specifically been denied the ability to log in to the system). The authenticating authority lets the Elements System know which authentication system in your institution should perform the authentication during user login.

Think of the authentication authority as the name of the system in your institution in which the user's login credentials are stored. The Elements System makes no guarantees to external systems that the usernames or authentication authority IDs of users will not be recycled or changed.

```
<api:object category="user" id="5" proprietary-id="3497" authenticating-authority="Internal" username="dwh" last-modified-when="2009-05-28T14:54:08.937+01:00" is-deleted="false" href="https://localhost:8091/publications-api/users/5" created-when="2005-07-28T15:46:18.377+01:00" type-id="1">...
```

The main object ID is the identifying information natively used by the API for users (as for all other objects) and is the ID you should generally use to directly fetch the user's data from the API. If you do not know the ID of a user, you can search for users by using the other two types of identifying user data (Proprietary ID or the authority/username combination), though you might return zero or multiple results, depending on your exact search settings and the distribution of usernames created by your institution.

The records belonging to a publication are often sourced from external data sources such as the Web of Science, PubMed, or arXiv. These records may have been assigned their own IDs by the external data sources.

Where such an ID exists and has been stored by the Elements System, the API will expose it in the details of the publication.

Do not confuse the external IDs of the records of a publication with the system ID of the publication itself.

```
<api:object category="publication" id="15" last-modified-when="2009-05-28T14:54:08.937+01:00" href="https://localhost:8091/publications-api/publications/15" created-when="2006-07-05T00:20:20.833+01:00" type-id="5">
  <api:ever-approved>true</api:ever-approved>
  <api:records>
    <api:record format="native" source-id="3" source-name="wos" source-display-name="Web of Science" id-at-source="A1986C172200006">...
```

Relationships

A relationship is a typed directed link from one object to another, which provides information about how the objects are related. No more than one relationship of each type may exist between any given pair of objects.

The relationship types provide the semantics of the Elements System's relational data storage.

An example relationship type is the "publication authorship" of a publication by a user. A relationship of this type is modelled in the system as a directed arrow of the appropriate type ID pointing from the publication to the user. This relationship indicates that the user is an author of the publication.

There are currently over 90 relationship types, each of which can be used to link an object of one specific category to an object of another specific category. For a comprehensive list of the relationship types, see the /relationship/types resource.

Some example relationships initially configured in the system are: publication derives from publication, publication authored by user, grant funds organisational structure, grant funds publication, activity related to concept, equipment used by user, user is PhD student of user, and many others.

Apart from having a type, a relationship may hold other information such as the dates between which the relationship was or is active.

Each relationship is assigned a unique integer ID by the Elements System.

```
<api:relationship id="138" type-id="8" href="https://localhost:8091/publications-api/relationships/138">
  <!--more data here if this is not a relationship reference-->
```

Relationship suggestions

The Elements system makes relationships suggestions to users of the system. Such suggestions are then either accepted (the suggestion is removed and a fully fledged relationship is created) or declined (and are remebered as such and not reoffered). You can see the full list of pending and declined relationship suggestions using the /suggestions/relationships/pending and /suggestions/relationships/declined resources.

In the normal course of events, an unwanted relationship suggestion is declined rather than deleted. It is possible to delete a relationship suggestion instead of declining it, but to do so may risk the suggestion being made by the system a second time. Declining a relationship suggestion is the recommended way of dismissing it.

If your API account has write permissions, you have the ability to decline or accept any relationship suggestion in the system, as well as reoffer existing relationships or declined suggestions for acceptance. See the individual operations on relationships suggestions and relationships for more information.

```
<api:relationship-suggestion id="138" type-id="8" href="https://localhost:8091/publications-
api/suggestions/relationships/138">
  <!--more data here if full detail was requested...-->
```

Research assessment

The Symplectic Elements system offers powerful research assessment functionality, and the API gives you access to the research assessment data stored in the Elements system. The “RA” resources cover the UK REF and the New Zealand PBRF research assessment exercises. The “HERDC” resources cover the Australian HERDC research assessment exercise.

HERDC return

A HERDC return is particular to a year, and is the top level organisational entity representing all nominations for a particular return.

HERDC nomination

A HERDC nomination represents the nomination of a research output for inclusion in a HERDC return. Associated data includes information about which publication/record was originally nominated, plus workflow status, associated HERDC-specific organisational group structure, and various HERDC-specific authorship data.

RA profile

An RA profile is the research assessment profile of a single Elements user, representing all of the information about the user relevant to your research assessment exercise, such as all of the user's nominated research outputs and declared special circumstances.

RA output

A research assessment output within the profile is created by the user when he/she nominates an existing research object in the system (such as a publication). At that point, the output is created from a snapshot of the data held in the object at the time of nomination, and thereafter enjoys an independent life within the user's research assessment profile. The output is represented by the API in a similar way to an object (although simplified).

If two users nominate the same publication (or other object), then two separate outputs are created in the two separate user research assessment profiles, and they remain independent of each other.

APPENDIX

Changes since the previous endpoint version (v3.7.16)

This section highlights the changes made since the v3.7.16 API compatibility endpoint.

The major changes are

i) the addition of the Teaching Activity category
ii) the addition of multiple records for a source
iii) the single ‘address’ element removed from person-list serialisation, use ‘addresses’ element instead
iv) the removal of the /objects resource
v) the removal of the /deleted/objects resource
vi) the removal of the /relationships-types resource
vii) the addition of the Identifier field type
viii) the addition of the Money field type
ix) “is-mesh” attribute removed from keyword field type
x) new maximum per page of 25 for full detail, view Pagination section for more details
xi) “journal-href” attribute removed from “api:issn” fields, use object level “api:journal" href instead
xii) for “repository-file”, the addition of “status” and “date-available” elements

The development of this functionality has necessitated some breaking changes to the REST URI structure of the API. See the section entitled “Upgrading from a v3.7.16 client” for the steps you may need to take to make a v3.7.16 client work with this API.

We have tried to keep such changes to a minimum while keeping options open for future development.

Decommisioned resources

Operation	Description
GET /objects	Use /{cats}
GET /deleted/objects	Use /deleted/{cats}
GET /relationship-types	Use /relationship/types

Changes to existing operations

Operation	v3.7.16 behaviour	v4.6behaviour
GET /{cats}	Teaching activities not returned	‘teaching-activity’ now an available category

Operation	v3.7.16 behaviour	v4.6behaviour
GET /{cats}	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}?detail=full	'address' element included in 'authors', 'is-mesh' attribute included in keywords	'address' element removed, use 'addresses' element instead. 'is-mesh' flag remove, use 'scheme' attribute on keyword element
GET /{cats}/{id}	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}?detail=full	'address' element included in 'authors', 'is-mesh' attribute included in keywords	'address' element removed, use 'addresses' element instead. 'is-mesh' flag remove, use 'scheme' attribute on keyword element
GET /{cats}/sources	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}/{cats2}	Teaching activities not an available category	'teaching-activity' now an available {cats} value and {cats2} value
GET /{cats}/{id}/{cats2}	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}/{id}/{cats2}?detail=full	'address' element included in 'authors', 'is-mesh' attribute included in keywords	Single 'address' element removed, use 'addresses' instead. 'is-mesh' flag remove, use 'scheme' attribute on keyword element
GET /{cats}/{id}/relationships	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}/relationships	Maximum per-page value 1000	Maximum per-page value 25 when detail level full

Operation	v3.7.16 behaviour	v4.6behaviour
GET /{cats}/{id}/relationships?detail=full	'address' element included in 'authors', 'is-mesh' attribute included in keywords	Single 'address' element removed, use 'addresses' instead. 'is-mesh' flag remove, use 'scheme' attribute on keyword element
GET /{cat}/records/{source}/{proprietary-id}	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cat}/records/{source}/{proprietary-id}?detail=full	'address' element included in 'authors', 'is-mesh' attribute included in keywords	Single 'address' element removed, use 'addresses' instead. 'is-mesh' flag remove, use 'scheme' attribute on keyword element
PUT /{cat}/records/{source}/{proprietary-id}	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}/suggestions/relationships/declined	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}/suggestions/relationships/declined	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}/{id}/suggestions/relationships/declined/{cats2}	Teaching activities not an available category	'teaching-activity' now an available {cats} value and {cats2} value
GET /{cats}/{id}/suggestions/relationships/declined/{cats2}	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}/{id}/suggestions/relationships/pending	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /{cats}/{id}/suggestions/relationships/pending	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}/{id}/suggestions/relationships/pending/{cats2}	Teaching activities not an available category	'teaching-activity' now an available {cats} value and {cats2}

Operation	v3.7.16 behaviour	v4.6behaviour
		value
GET /{cats}/{id}/suggestions/relationships/pending/{cats2}	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /{cats}/types	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /deleted/{cats}	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /deleted/{cats}/{id}	Teaching activities not an available category	'teaching-activity' now an available {cats} value
GET /ra-portfolios	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /relationships	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /suggestions/relationships/declined	Maximum per-page value 1000	Maximum per-page value 25 when detail level full
GET /suggestions/relationships/pending	Maximum per-page value 1000	Maximum per-page value 25 when detail level full

Upgrading a v3.7.16 client

This section walks you through the changes you must make, or reviews you must perform to upgrade your application from working with a v3.7.16 endpoint to work with the v4.6 endpoint in exactly the same way.

The steps are deliberately minimal and do not encourage you to make use of newer functionality if they do not have to. You may not need to make all the changes if the difference in API behaviour since v3.7.16 does not affect your use case.

i) When viewing a resource with the ?detail=full parameter, a max page size of 25 is enforced
ii) Grant amount and currency fields from the v3.7.16 API and earlier are now defined as a single amount field of type money in the 4.6 API.

Request changes

Wherever you have called an operation in the first column, you **must** rewrite your call as per the description in the second column. Otherwise you will either prompt an error response, or see unexpected results in the response.

v3.7.16 operation(s)	Equivalent v4.6 operation(s)
GET /objects	GET /{cats}
GET /deleted/objects	GET /deleted/{cats}
GET /relationship-types	GET /relationship/types

Changes to previous endpoint XML

The following colour-codes represent changes to parts of the v3.7.16 XML schema in v4.6:

Changes: grey

Additions: yellow

Deletions: cyan

The XML schema is now also self-documented, using standard XSD support for schema annotations.

```
<xs:complexType name="data-value">
  <xs:sequence>
    <xs:choice>
      <xs:element name="addresses">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="address" type="api:address"
              minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
```

```
      </xs:element>
      <xs:element name="boolean" type="xs:boolean"/>
      <xs:element name="date" type="api:date"/>
      <xs:element name="decimal" type="xs:decimal"/>
      <xs:element name="funding-acknowledgements" type="api:funding-acknowledgements"/>
      <xs:element name="identifiers">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="identifier" type="api:identifier"
minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="integer" type="xs:int"/>
      <xs:element name="items">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="item" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="keywords">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="keyword" type="api:keyword"
              minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="money" type="api:money"/>
      <xs:element name="pagination" type="api:pagination"/>
      <xs:element name="people">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="person" type="api:person"
            minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="text" type="xs:string"/>
    </xs:choice>
    <xs:element name="links" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="link" maxOccurs="unbounded">
            <xs:complexType>
              <xs:attribute name="type" type="api:link-type"/>
              <xs:attribute name="href" type="xs:anyURI"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="journal-href" type="xs:anyURI" use="optional"/>
  <xs:attribute name="type" type="api:field-type" use="required"/>
</xs:complexType>
```

```
<xs:complexType name="import-native-record">
  <xs:sequence>
    <xs:element name="field" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:choice>
          <xs:element name="addresses">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="address" type="api:address"
 minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="boolean" type="xs:boolean"/>
          <xs:element name="date" type="api:date">
          <xs:element name="decimal" type="xs:decimal"/>
          <xs:element name="funding-acknowledgements" type="api:funding-
acknowledgements"/>
          <xs:element name="identifiers">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="identifier" type="api:identifier"
 minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="integer" type="xs:int"/>
          <xs:element name="items">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="item" type="xs:string"
 minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="keywords">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="keyword" type="api:keyword"
 minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="money" type="api:money"/>
          <xs:element name="pagination" type="api:pagination"/>
          <xs:element name="people">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="person" type="api:person"
 minOccurs="0" maxOccurs="unbounded"/>
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="text" type="xs:string"/>
        </xs:choice>
        <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
```

```
  </xs:sequence>
</xs:complexType>

<xs:complexType name="keyword">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="is-mesh" type="xs:boolean" use="optional"/>
      <xs:attribute name="scheme" type="xs:string" use="optional"/>
      <xs:attribute name="percentage" type="xs:int" use="optional"/>
      <xs:attribute name="origin" type="api:label-origin" use="optional"/>
      <xs:attribute name="source" type="xs:string" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>


<xs:simpleType name="category">
  <xs:restriction base="xs:string">
    <xs:enumeration value="activity"/>
    <xs:enumeration value="concept"/>
    <xs:enumeration value="equipment"/>
    <xs:enumeration value="grant"/>
    <xs:enumeration value="org-structure"/>
    <xs:enumeration value="project"/>
    <xs:enumeration value="publication"/>
    <xs:enumeration value="teaching-activity"/>
    <xs:enumeration value="user"/>
  </xs:restriction>
</xs:simpleType>
```

```
<xs:complexType name="search-field">
  <xs:sequence>
    <xs:element name="value-types">
      <xs:complexType>
        <xs:sequence>
          <xs:element maxOccurs="unbounded" name="value-type">
            <xs:complexType>
              <xs:attribute name="name" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="composite"/>
                    <xs:enumeration value="text"/>
                    <xs:enumeration value="integer"/>
                    <xs:enumeration value="number"/>
                    <xs:enumeration value="date-time"/>
                    <xs:enumeration value="boolean"/>
                    <xs:enumeration value="url"/>
                    <xs:enumeration value="isbn-10"/>
                    <xs:enumeration value="isbn-13"/>
                    <xs:enumeration value="issn"/>
                    <xs:enumeration value="doi"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
```

```
      </xs:complexType>
    </xs:element>
    <xs:element name="supported-operators">
      <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
          <xs:element maxOccurs="unbounded" name="supported-operator">
            <xs:complexType>
              <xs:attribute name="name" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="equal"/>
                    <xs:enumeration value="not-equal"/>
                    <xs:enumeration value="less-than"/>
                    <xs:enumeration value="less-than-equal"/>
                    <xs:enumeration value="greater-than"/>
                    <xs:enumeration value="greater-than-equal"/>
                    <xs:enumeration value="similar"/>
                    <xs:enumeration value="not-similar"/>
                    <xs:enumeration value="in"/>
                    <xs:enumeration value="not-in"/>
                    <xs:enumeration value="is-empty"/>
                    <xs:enumeration value="is-not-empty"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
              <xs:attribute name="symbol" use="required">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:enumeration value="="/>
                    <xs:enumeration value="!="/>
                    <xs:enumeration value="&lt;"/>
                    <xs:enumeration value="&lt;="/>
                    <xs:enumeration value="&gt;"/>
                    <xs:enumeration value="&gt;="/>
                    <xs:enumeration value="~"/>
                    <xs:enumeration value="!~"/>
                    <xs:enumeration value="in"/>
                    <xs:enumeration value="not in"/>
                    <xs:enumeration value="is empty"/>
                    <xs:enumeration value="is not empty"/>
                  </xs:restriction>
                </xs:simpleType>
              </xs:attribute>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="context" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="activity"/>
        <xs:enumeration value="concept"/>
        <xs:enumeration value="element"/>
        <xs:enumeration value="equipment"/>
        <xs:enumeration value="grant"/>
        <xs:enumeration value="org-structure"/>
```

```
        <xs:enumeration value="project"/>
        <xs:enumeration value="publication"/>
        <xs:enumeration value="relationship"/>
        <xs:enumeration value="teaching-activity"/>
        <xs:enumeration value="user"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="identifier">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="scheme" type="api:identifier-scheme" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="identifier-scheme">
  <xs:restriction base="xs:string">
    <xs:enumeration value="pmc" />
    <xs:enumeration value="dais" />
    <xs:enumeration value="arxiv" />
    <xs:enumeration value="pubmed" />
    <xs:enumeration value="nihms" />
    <xs:enumeration value="isidoc" />
    <xs:enumeration value="researcherid" />
    <xs:enumeration value="orcid" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="person">
  <xs:sequence>
    <xs:element name="last-name" type="xs:string"/>
    <xs:element name="initials" type="xs:string"/>
    <xs:element name="address" type="api:address" minOccurs="0"/>
    <xs:element name="addresses" type="api:addresses" minOccurs="0"/>
    <xs:element name="email-address" type="xs:string" minOccurs="0"/>
    <xs:element name="proprietary-id" type="xs:string" minOccurs="0">
    <xs:element name="identifiers" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="identifier" type="api:identifier"
minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>


<xs:simpleType name="field-type">
  <xs:restriction base="xs:string">
    <xs:enumeration value="address-list"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="decimal"/>
    <xs:enumeration value="funding-acknowledgements"/>
```

```
    <xs:enumeration value="integer"/>
    <xs:enumeration value="identifier-list"/>
    <xs:enumeration value="keyword-list"/>
    <xs:enumeration value="list"/>
    <xs:enumeration value="money"/>
    <xs:enumeration value="pagination"/>
    <xs:enumeration value="person-list"/>
    <xs:enumeration value="text"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="money">
  <xs:simpleContent>
    <xs:extension base="xs:decimal">
      <xs:attribute name="iso-currency" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="AUD" />
            <xs:enumeration value="CAD" />
            <xs:enumeration value="EUR" />
            <xs:enumeration value="GBP" />
            <xs:enumeration value="JPY" />
            <xs:enumeration value="NZD" />
            <xs:enumeration value="USD" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<xs:complexType name="record">
  <xs:sequence>
    <xs:element name="citation-count" type="xs:int" minOccurs="0"/>
    <xs:element name="verification-status" type="api:verification-status" minOccurs="0"/>
    <xs:element name="verification-comment" type="xs:string" minOccurs="0"/>
    <xs:element name="native" type="api:native-record"/>
  </xs:sequence>
  <xs:attribute name="format" type="api:record-format" use="required"/>
  <xs:attribute name="source-id" type="xs:int" use="required"/>
  <xs:attribute name="source-name" type="xs:string" use="required"/>
  <xs:attribute name="source-display-name" type="xs:string" use="required"/>
  <xs:attribute name="id-at-source" type="xs:string" use="required"/>
  <xs:attribute name="is-preferred-record" use="optional"/>
    <xs:simpleType>
      <xs:restriction base="xs:boolean">
        <xs:pattern value="true"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:complexType>

<xs:complexType name="relationship">
  <xs:sequence>
    <xs:element name="is-visible" type="xs:boolean" minOccurs="0"/>
    <xs:element name="is-favourite" type="xs:boolean" minOccurs="0"/>
    <xs:element name="preferred-source-id" type="xs:int" minOccurs="0"/>
    <xs:element name="preferred-source-name" type="xs:string" minOccurs="0"/>
```

```
    <xs:element name="date-1" type="xs:date" minOccurs="0"/>
    <xs:element name="date-2" type="xs:date" minOccurs="0"/>
    <xs:element name="related" minOccurs="0" maxOccurs="2">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="object" type="api:object"/>
        </xs:sequence>
        <xs:attribute name="direction" type="api:relationship-direction" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="id" type="xs:int" use="required"/>
  <xs:attribute name="type-id" type="xs:int" use="required"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
  <xs:attribute name="href" type="xs:anyURI" use="required"/>
</xs:complexType>

<xs:complexType name="object">
  <xs:sequence minOccurs="0">
    <xs:element name="merge-history" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="merge" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
              <xs:all>
                <xs:element name="merged-object">
                  <xs:complexType>
                    <xs:attribute name="id" type="xs:int" use="required"/>
                    <xs:attribute name="href" type="xs:anyURI" use="required"/>
                  </xs:complexType>
                </xs:element>
                <xs:element name="merged-into-object">
                  <xs:complexType>
                    <xs:attribute name="id" type="xs:int" use="required"/>
                    <xs:attribute name="href" type="xs:anyURI" use="required"/>
                  </xs:complexType>
                </xs:element>
              </xs:all>
              <xs:attribute name="when" type="xs:dateTime" use="required">
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="ever-approved" type="xs:boolean" minOccurs="0">
    <xs:element name="reporting-date-1" type="xs:date" minOccurs="0"/>
    <xs:element name="reporting-date-2" type="xs:date" minOccurs="0"/>
    <xs:element name="is-current-staff" type="xs:boolean" minOccurs="0"/>
    <xs:element name="photo" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="is-academic" type="xs:boolean" minOccurs="0"/>
    <xs:element name="title" type="xs:string" minOccurs="0"/>
    <xs:element name="initials" type="xs:string" minOccurs="0"/>
    <xs:element name="last-name" type="xs:string" minOccurs="0"/>
    <xs:element name="first-name" type="xs:string" minOccurs="0"/>
```

```
    <xs:element name="email-address" type="xs:string" minOccurs="0"/>
    <xs:element name="known-as" type="xs:string" minOccurs="0"/>
    <xs:element name="suffix" type="xs:string" minOccurs="0"/>
    <xs:element name="primary-group-descriptor" type="xs:string" minOccurs="0"/>
    <xs:element name="organisation-defined-data" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="field-number" type="xs:int" use="required"/>
            <xs:attribute name="field-name" type="xs:string" use="required"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="repository-items" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="repository-item" minOccurs="0">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="public-url" type="xs:string" minOccurs="0"/>
                <xs:element name="content-file-count" type="xs:int" minOccurs="0"/>
                <xs:element name="licence-file-count" type="xs:int" minOccurs="0">
                <xs:element name="first-file-uploaded-
when" type="xs:dateTime" minOccurs="0"/>
                <xs:element name="first-licence-granted-
when" type="xs:dateTime" minOccurs="0"/>
                <xs:element name="status" type="xs:string" minOccurs="1"/>
                <xs:element name="repository-files" minOccurs="0">
                  <xs:complexType>
                    <xs:sequence>
                      <xs:element name="repository-
file" minOccurs="0" maxOccurs="unbounded">
                        <xs:complexType>
                          <xs:sequence>
                            <xs:element name="name" type="xs:string" minOccurs="0"/>
                            <xs:element name="public-url" type="xs:string" minOccurs="0"/>
                            <xs:element name="mime-type" type="xs:string" minOccurs="0"/>
                            <xs:element name="version" type="xs:string" minOccurs="0"/>
                            <xs:element name="date-
available" type="xs:dateTime" minOccurs="0"/>
                          </xs:sequence>
                          <xs:attribute name="type" type="xs:string" use="required"/>
                          <xs:attribute name="last-modified-
when" type="xs:dateTime" use="required"/>
                        </xs:complexType>
                      </xs:element>
                    </xs:sequence>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="repository-id" type="xs:int" use="required"/>
              <xs:attribute name="repository-name" type="xs:string" use="required"/>
            </xs:complexType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
    <xs:element name="records" minOccurs="0"/>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="record" type="api:record" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="fields" minOccurs="0">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="field" type="api:field-
value" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="all-labels" minOccurs="0" type="api:data-value"/>
    <xs:element name="journal" type="journal" minOccurs="0"/>
    <xs:element name="relationships" minOccurs="0">
      <xs:complexType>
        <xs:attribute name="href" type="xs:anyURI" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="category" type="api:category" use="required"/>
  <xs:attribute name="id" type="xs:int" use="required">
  <xs:attribute name="proprietary-id" type="xs:string" use="optional"/>
  <xs:attribute name="authenticating-authority" type="xs:string" use="optional"/>
  <xs:attribute name="username" type="xs:string" use="optional"/>
  <xs:attribute name="last-modified-when" type="xs:dateTime" use="required"/>
  <xs:attribute name="href" type="xs:anyURI" use="required"/>
  <xs:attribute name="created-when" type="xs:dateTime" use="required"/>
  <xs:attribute name="type-id" type="xs:int" use="required"/>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:complexType>
```

Troubleshooting

Connection Reset (Firefox: "The connection was reset"; Internet Explorer: "cannot view the webpage")

This response is made by the operating system when you access a secure HTTPS endpoint to which your IT support/application administrator has not assigned an SSL certificate.

Contact your administrator and request that your IT staff binds an SSL certificate to the port to which you are trying to connect. Instructions for this process are supplied in the 'API Administrator Guide'.

HTTP 400 - Bad Request

1. Check the response for ATOM content. If ATOM content is returned, see the contained API error code and human-readable explanation of the error.

2. Check that the URL of the operation you are attempting is correctly formed. For example, the URL must not contain a double-slash except immediately after the scheme (http://).

3. If you are using a PUT or POST based operation, check that your request includes a "Content-Type: text/xml" HTTP header.

4. Check that any XML that you are providing is well-formed. You can use online tools to help you with this, for example http://validator.w3.org/. Always use library code to generate your XML, as discussed in Best Practices.

5. Check that any XML that you are providing is validated by the API schema. You can use online tools to help you with this, for example http://www.xmlforasp.net/SchemaValidator.aspx. This is the most common way to cause a Bad Request response. Sometimes, an online validator like this will suggest that the problem is deeply nested in your XML when the actual problem is a missing or incorrect namespace on your document (root) element.

6. If the operation you are attempting offers a "validate" parameter, repeat the request setting this value to "true", and read any resulting error message. Do not use this parameter as a matter of course in live production code, as it causes a significant performance hit on the Elements System and is only provided to temporarily help you with your debugging.

HTTP 401 - Unauthorized

You have not correctly (or at all) submitted your credentials with your request, or your credentials were invalid.

Make sure that you are supplying a "WWW-Authenticate" header with every request. The value of the header should be the string "Basic " (including the trailing space character) concatenated with the

base 64 encoded value of the UTF-8 encoded value of the string "username:password" (replace "username" with your username, and "password" with your password).

For example, the username "foo" with the password "bar" should be submitted using the following HTTP header:

```
WWW-Authenticate: Basic Zm9vOmJhcg==
```

The steps to achieve this are

a) "username:password" becomes "foo:bar"
b) "foo:bar" encoded using UTF-8 becomes the byte array [0x66, 0x6f, 0x6f, 0x3a, 0x62, 0x61, 0x72].
c) The byte array [0x66, 0x6f, 0x6f, 0x3a, 0x62, 0x61, 0x72] represented in base 64 is the string "Zm9vOmJhcg==".

If you are correctly attaching your credentials and still seeing this response, then either your credentials are invalid or you are accessing the API from a machine whose IP address has not been registered with the Elements system. Contact your system administrator, providing him or her with your machine's IP address.

HTTP 403 - Forbidden

Check with the system administrator that the account corresponding to the credentials you are using has the required permissions for the operation you are attempting. If you are attempting an operation with anything except the "GET" HTTP method, your account requires read/write access to the Elements database. See the GET /my-account operation to review the rights that your account has been assigned.

Timeout connecting to the API

1. Use a telnet client from the machine on which your client is running to verify that the target port is open on the target machine. If it is, then the API is available for connections. Contact your network support for more help.

2. Check with your system administrator that the API has been configured by your institution to run on the port you are attempting to access.

3. Check with your system administrator that the API has been switched on.

4. Contact your system administrator to have them check using a telnet client on the machine on which the API is running to verify that the target port is open when accessed from that machine. If so, the problem is a firewall or network configuration problem. Contact your local network support for assistance.

Timeout while waiting for API operation to complete

1. Log the full details of the requests that are causing the timeout to understand the timing of the requests you are making and the amount of time the requests are taking to respond.

2. Are you executing multiple concurrent requests to a plan? If so, scale back the concurrency and introduce sleep time between receiving responses and issuing new requests.

3. Are you executing requests at a varying rate determined by an external stimulus such as incoming requests from the internet? Do the timeouts occur at periods of high demand? If so, the number of requests per second is reaching too high a value for the Elements system, and you need to reengineer your solution to use a caching approach. See the Best Practices section of the Appendix.

4. Can you verify that the timeouts consistently occur for some supplied parameters, but not for others, when requesting the same operation? If so, contact the Symplectic support desk.

5. Are you executing your requests at the same time as a scheduled system process? If you find your performance problems occur at regular intervals, this is likely the case. Liaise with the system administrator to find an alternative time to run your requests against the API.

6. Are you building in any sleep time between receiving a response and the issuing of the next request? Increase this wait time and monitor any change in the incidence of timeouts.