

## Rustici Software : Web Service API Documentation

---

This page last changed on May 28, 2009 by [david.ells](#).

### Overview

This page exists to document the web services API. It differs from other service listings found [nearby](#) in the wiki, in that this should roughly translate into a public API document for interacting with a web services instance. The web services, in turn, use the "internal" service interface (in the linked documentation) to perform their tasks. (Similarly, a fully written client library would use the documentation here to interact with web services on behalf of a user, and ideally provide the same "internal" service interface to the client. This makes the client library a *consumer of the web services* and a *provider of the services* (to client code).

All web service calls (through the current interface) are of the form [web service instance]/api?method=[service.prefix].[methodName]&appid=[your app id](*&any=other&methodparams=here*)(*&security=params*)

For secured services (which will likely be all services offered), the security params include the ts param (a UTC timestamp of the form yyyyMMddHHmmss) and the sig param (an MD5 hash signature of the method call, detailed below)

Authentication for web service calls is largely modeled off of the flickr model, which can be found [here](#). In summary, each http request sent to a secured web services instance must contain the following parameters to pass the security check:

- appid : Your application id, acquired upon registration with the IKI services website
- ts : A timestamp in the UTC timezone of the form yyyyMMddHHmmss (ex. 20081217223530)
- sig : A signature created by hashing the specifics of a request with your secret key

The signature ("sig" param) is created using the following algorithm:

- Every parameter to be included on the query string of a call (every parameter except for "sig") is sorted by key name (e.g. foo=1&bar=2 is sorted as bar=2, foo=1)
- The sorted key value pairs are concatenated, with any query string specific characters removed (e.g. bar=2&foo=1 -> bar2foo1)
- This "serialized request string" is prepended with your secret key (e.g. BIGSECRETbar2foo1)
- An MD5 hash is calculated of the *UTF-8 encoding* of the combined string, and then converted to a hex string (e.g. md5hash(BIGSECRETbar2foo1) -> 7D6AED5AD524170A018D4AC8FFE0693C)

The resulting hex string is the signature for the method call and is attached as the sig parameter (e.g. api?method=[method]&appid=[appid]&ts=[timestamp]&sig=7D6AED5AD524170A018D4AC8FFE0693C)  
The server will perform the same operations listed above and allow access if the signatures match.

The timestamp sent in the request must match the time on the server (within an expiration limit). In this way, every service request (over time) is uniquely signed, even if the same parameters are used in the call.

Every web service call should result in a formatted response. The current implementation defaults to using a rest XML format similar to the flickr api. A typical "successful" response will be of this form:

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="ok">
  [xml payload]
</rsp>
```

And a typical "error" response (usually caused by an error or exception in performing the requested operation) should have the form:

```
<?xml version="1.0" encoding="utf-8" ?>
<rsp stat="fail">
  <err code="[error code here]" msg="[error message here]" />
  [*potentially <err code="[inner exception error code]" msg="[inner exception message]" />]
  [*optionally
  <errordetails>
    <stacktrace>
      <tracetext>text</tracetext>...
    </stacktrace>
  </errordetails>]
</rsp>
```

The error return can optionally include an errordetails element that includes extra diagnostic information (like a stack trace). This is enabled by a configuration variable in the web service core code. Perhaps it should be a common optional parameter to pass to any method...

There are a number of common error codes that can be returned by all available methods, which are as follows:

Common error codes:

- Security Errors
  - 100 : A general security error has occurred, i.e. the security check has failed.
  - 101 : The required appid parameter is missing.
  - 102 : The required appid is invalid.
  - 103 : The required sig parameter (request signature) is missing.
  - 104 : The sig parameter is invalid (signature sent in the request does not match signature generated on the server)
  - 105 : The required ts parameter (the UTC timestamp) is missing
  - 106 : The required ts parameter is invalid (needs to be of the form yyyyMMddHHmmss)
  - 107 : The passed ts parameter has expired (timestamp sent in request must be within certain amount of time within when server fulfills the request)
  - 108 : An error has occurred in accessing the back end storage for authentication
  
- Service Errors
  - 200 : Invalid service call (the method parameter is missing or incorrect)
  - 201 : Service unavailable
  - 202 : Method not found
  - 203 : A required parameter for the called method is missing from the request
  - 204 : An invalid format parameter is specified (for now format is ignored, only XML is returned)
  - 205 : Invalid method parameter value

Error codes 0-99 specify errors that have occurred during processing of a web service call. Error code 99 is reserved to note "general errors", which refer to exceptions that are thrown in processing that are not handled by the web service in a more specific way.

In the service call specifications below, the example response will show just the xml payload that is wrapped in a successful <rsp> element as shown above. Similarly the error codes and messages noted for each method will be wrapped in the common error response format shown above.

## API Listing

### Debug Service (rustici.debug)

This unsecured service is to provide a very basic set of simple method calls to check the status of a running web service instance. Some of the calls present in this service, or the entire service, should be disabled for production deployments of the web services instance.

#### ping

- Semantics: This method simply responds to the user. It's purpose is to have a simple way of testing the underlying mechanisms for delivering the web services (i.e. the web service "core") and making sure the instance is up and running
- Required Arguments: none
- Optional Arguments: none
- Example call: [web service instance]/api?method=rustici.debug.ping
- Example response:

```
<pong />
```

#### getTime

- Semantics: This method returns the time on the running instance, in (UTC/GMT) timezone in the form 'yyyyMMddHHmmss'. This format is used for the timestamp that is part of the security mechanism.
- Required Arguments: none
- Optional Arguments: none
- Example call: [web service instance]/api?method=rustici.debug.getTime
- Example response:

```
<currenttime tz="UTC">20081217152345</currenttime>
```

### Upload Service (rustici.upload)

This secured service provides a way to upload (over HTTP) files that can be used in subsequent web service calls. Primarily, this service exists for uploading course zip files that can be used in a subsequent call to `rustici.course.importCourse` (with the optional "path" parameter). Each application targeting the web services will be granted it's own upload area, and every upload operation must include the `appid` parameter. Under the application's upload area, specific "permission domains" can be created. These permission domains allow a consuming application to isolate disparate files, and permission domains can be specified in programmatic requests for FTP access. Thus a consuming application can create various upload areas specific to underlying users or groups within their application, but they also hold the responsibility of controlling access (atleast via web service calls, not FTP) to these domains under their upload area.

## uploadFile

- **Semantics:** This method expects the incoming HTTP request to be a POST of multipart (i.e. HTTP form w/ input of type file) data. Every other parameter (other than the file data) must be specified on the querystring. If the optional domain parameter is included, the file will be placed under this domain within the application's upload area. This method will return a relative path to the file (relative to the root of your specific application's upload area). If the optional `redirecturl` parameter is specified, the browser sending the request will be redirected to the given url, appending the following querystring parameters: a "success" parameter noting success/failure, and if successful, a "location" parameter that specifies the relative location of the uploaded file (relative to the root of your application). If the call failed, the url will be appended with an "errcode" parameter specifying the error code, and a "msg" parameter specifying the error message. This allows developers to direct the user automatically to another page when a form is submitted in a browser.
- **Required Arguments:**
  - `appid` : Your application id
- **Optional Arguments:**
  - `pd` : The "permission domain" in which to place the uploaded file. If unspecified, "default" is used.
  - `redirecturl` : The url that a browser should be redirected to after the operation completes.
- **Example call:** An example call is best illustrated with a simple http form:

```
<form id="uploadtester" method="post"
      action="[web service
instance]/api?method=rustici.upload.uploadFile&appid=myappid&pd=testdomain"
      enctype="multipart/form-data">
  <input type="file" name="filedata" size="40" /> <br />
  <input type="submit" value="Send" />
</form>
```

- **Example response:**

```
<location>testdomain/Photoshop_None.zip</location>
```

- **Error codes:**
  - 1 : Error in processing file upload request that was sent
  - 2 : File upload caused an IO error
  - 3 : No file data was found in the request
  - 4 : The file uploaded did not have the required extension (some instances may force uploads of only a certain type)
  - 5 : Some other error occurred in uploading the file

## listFiles

- Semantics: A call to this method will return a listing of files that have been uploaded using the given appid, and optionally, under the given permission domain. Permission domains are briefly explained in the semantics section of the uploadFile call listed above. (TODO: Link to more detailed information about permission domains and security tags, maybe that text belongs w/ the ftp service?)
- Required Arguments:
  - appid : Your application id
- Optional Arguments:
  - pd : The "permission domain" from which to retrieve the list of files to return
- Example call: [web service instance]/api?method=rustici.upload.listFiles&appid=myappid(&pd=testdomain)
- Example response:

```
<dir name="testdomain">
  <file name="LMSTestPackage_API.zip" size="106866" modified="20090109194600"/>
  <file name="Photoshop_Compency.zip" size="552868" modified="20081216200658"/>
  <file name="Photoshop_KnowledgePaced.zip" size="553332" modified="20090108215221"/>
</dir>
```

- Error codes:
  - 1 : The permission domain specified does not exist

## deleteFiles

- Semantics: A call to this method will attempt to delete each file specified in the file parameter, belonging to the given permission domain, or the default permission domain if no domain is specified.
- Required Arguments:
  - appid : Your application id
  - file : The file(s) to be deleted. Note that this parameter can have multiple values, in which case, each file specified will be deleted (or attempted to be deleted, that is).
- Optional Arguments:
  - pd : The permission domain in which the specified files exist. If not specified, use the default domain.
- Example call: [web service instance]/api?method=rustici.upload.deleteFiles&appid=myappid&pd=testdomain&file=Photoshop\_Remediation.z
  - \*Note that it could be fairly easy to exceed the maximum length of a query string based GET request with a long list of files, and hence the POST action should be used in those cases
- Example response:

```
<results>
  <result file="Photoshop_Remediation.zip" deleted="true"/>
  <result file="NotReallyThere.zip" deleted="false" />
</results>
```

- Error Codes:
  - Any errors in deleting the named files will just report a deleted="false" for that file.

## Course Service (rustici.course)

This secured service allows for many operations related to courses, including importing new courses and retrieving information about imported courses.

## importCourse

- **Semantics:** Posting file data in a multipart HTTP request to this method will result in the upload and import of the file, and will return messages about import on success. If the optional path parameter is specified, the service will try to import the course named by the path parameter by looking for this path under the uploads area specific to the given appid. This variant of the method should be used when the file to be imported has already been uploaded, either through a web service call to `rustici.upload.uploadFile`, or through FTP access. If the optional `redirecturl` parameter is specified, the browser sending the request will be redirected to the given url, appending the following querystring parameters: an "success" parameter noting success/failure of the import, a "title" parameter containing the title of the imported course, and a "msg" parameter containing the accompanying message for the import. If the call failed, the url will be appended with an "errcode" parameter specifying the error code, and a "msg" parameter specifying the error message. This allows developers to direct the user automatically to another page when a form is submitted in a browser.
- **Required Arguments:**
  - `appid` : Your application id
  - `courseid` : The id used to identify this course
- **Optional Arguments:**
  - `path` : The relative path (rooted at your specific appid's upload area) where the zip file for importing can be found
  - `itemid`: The item identifier of a specific learning object to import. This allows for the import of an aggregation or a specific SCO/asset within an organization.
  - `pd` : An permission domain to associate this course with, for ftp access service (see ftp service below). If the domain specified does not exist, the course will be placed in the default permission domain
  - `redirecturl` : Optionally, instead of returning xml, redirect to the given page, appending parameters detailing success or failure of the call
- **Example call:** [web service instance]/api?method=rustici.course.importCourse&appid=myappid&courseid=course003(&path=default/CourseC (or HTTP multipart post data if no path specified)
- **Example response:**

```
<importresult successful="true">
<title>Photoshop Example -- Competency</title>
<message>Import Successful</message>
[possibly <parserwarnings>
  <warning>[warning text]</warning>
</parserwarnings> ]
</importresult>
```

- **Error codes:**
  - 1 : A course with the given courseid already exists (to create a new version of a course, use `versionCourse`)
  - 2 : The file specified in the optional path parameter can't be found

## **versionCourse**

- **Semantics:** This method is used to create a new version of an existing course. A course must first exist in order to be specified (by courseid) in `versionCourse`. Other than this difference, the details of both the request and response match the `importCourse` call outlined above
- **Required Arguments:**
  - Same as `importCourse` above
- **Optional Arguments:**
  - Same as `importCourse` above
- **Example call:** [web service instance]/api?method=rustici.course.importCourse&appid=myappid&courseid=course003(&path=default/CourseC

(or HTTP multipart post data if no path specified)

- Example response:
  - Same as importCourse above
- Error codes:
  - 1 : The course specified by the given courseid can not be found
  - 2 : The file specified in the optional path parameter can't be found

## getAssets

- Semantics: This method provides a simple mechanism to download files for a given course. If no error occurs during processing of the call, this method will not return any XML response, but rather an application/octet-stream response containing the data of the requested course assets. This call provides a way to quickly retrieve important files like the imsmanifest.xml or metadata files, or many files at once, returned in the form of a zip file. If the optional path parameter is not specified, all the assets for the course will be returned in the zip file.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course under which the assets should be found.
- Optional Arguments:
  - path : The (relative) file path to a specific asset. The path is relative to the course root. If this parameter has only a single value, just the one file will be returned. If this parameter has multiple values, each of the assets specified will be packaged together into a zip file. If this parameter is not specified, all the assets for the course will be returned in the zip file. The structure of the zip file will have the courseid as the root, and each file will be at the relative path specified underneath that root. (If no path specified, the structure will exactly resemble the structure returned in the call to getFileStructure.)
  - versionid : The version of the package which will be used. If absent, use the most recent version.
- Example call: [web service instance]/api?method=rustici.course.getAsset&appid=myappid&courseid=course003(&versionid=2&path=images)
- Example response:
  - Success will result in a file download, while an error will return error details in the common format

## updateAssets

- Semantics: This method is very similar to versionCourse above, except that a new version of the course is **not** created, only the assets of the course are updated. Files found in the zip file, which is sent through the request or specified by the optional path parameter, will be overlayed on top of the files belonging to the course specified by courseid (and versionid). Note however that the existing manifest file will **not** be overwritten.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for deletion
- Optional Arguments:
  - versionid : The version of the package which will be updated. If absent, use the most recent version.
- Example call: [web service instance]/api?method=rustici.course.updateAssets&appid=myappid&courseid=course003(&versionid=[version number]&path=default/Course003.zip) (or HTTP multipart post data if no path specified))
- Example response:

```
<success />
```

- Error codes:
  - 1 : The course specified by the given courseid cannot be found
  - 2 : The file specified in the optional path parameter can't be found
  - 3 : Internal processing of the assets file caused an IO exception

## deleteCourse

- Semantics: A call to this method will cause the course specified by the courseid parameter (belonging to the client named by appid) to be deleted. A call can optionally include a versionid parameter which will name a specific version of the course to be deleted. If absent, all versions of the package will be deleted.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for deletion
- Optional Arguments:
  - versionid : The version of the package which will be deleted. If absent, delete all versions of the package. If "latest", delete just the most recent version of the package.
- Example call: [web service instance]/api?method=rustici.course.deleteCourse&appid=myappid&courseid=course003(&versionid=[version number] OR versionid=all)
- Example response:

```
<success />
```

- Error codes:
  - 1 : The course specified by the given courseid cannot be found
  - 2 : Deleting the files associated with this course caused an internal security exception

## getFileStructure

- Semantics: This method returns xml that represents the file structure of the course files belonging to the course named by the courseid parameter, along with some very basic metadata about the files, such as file length and last modified date. All dates shown are of the format yyyyMMddHHmmss and in the UTC timezone. Including the optional versionid parameter will target a specific version of the course for this call.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course
- Optional Arguments:
  - versionid : The specific version of the course that you wish to act upon. If absent, use the most recent version.
- Example call: [web service instance]/api?method=rustici.course.getFileStructure&appid=myappid&courseid=course003(&versionid=[version number])
- Example response:

```
<dir name="photoshop_competency">  
  <file name="adlcp_vlp3.xsd" size="2846" modified="20090108210616"/>  
  <file name="adlnav_vlp3.xsd" size="2109" modified="20090108210616"/>
```

```

<file name="adlseq_vlp3.xsd" size="2649" modified="20090108210616"/>
<file name="datatypes.dtd" size="6357" modified="20090108210616"/>
<file name="Exam.htm" size="20073" modified="20090108210616"/>
<dir name="images">
  <file name="37.gif" size="70" modified="20090108210616"/>
  <file name="72.gif" size="169" modified="20090108210616"/>
  <file name="Addition.gif" size="136" modified="20090108210616"/>
  [...]
  <file name="ZoomToolIcon.gif" size="142" modified="20090108210616"/>
</dir>
[...]
<file name="XMLSchema.dtd" size="16018" modified="20090108210616"/>
</dir>

```

- Error Codes:
  - 1 : Could not find the given course specified by courseid (and versionid) belonging to the given appid

## deleteFiles

- Semantics: A call to this method will attempt to delete each file specified by the relative path named in the path parameter. The path specified is relative to the root of the course files belonging to the course named by the courseid parameter. The method returns a list of results for each path, and whether the deletion was successful.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course whose files are named for deletion
  - path : The relative path (relative to the course file root) to the file(s) to be deleted. Note that this parameter can have multiple values, in which case, each file specified will be deleted (or attempted, that is).
- Optional Arguments:
  - versionid : The specific version of the course that you wish to act upon. If absent, use the most recent version.
- Example call: [web service instance]/api?method=rustici.course.deleteFiles&appid=myappid&courseid=course003&path=image1.jpg(&path=
  - \*Note that it could be fairly easy to exceed the maximum length of a query string based GET request with a long list of files, and hence the POST action should be used in those cases
- Example response:

```

<results>
  <result path="images/birdfeeder.jpg" deleted="true"/>
  <result path="images/doesnotexist.jpg" deleted="false"/>
</results>

```

- Error Codes:
  - 1 : Course named by courseid not found
  - Any errors in deleting the named files will just report a deleted="false" for that file.

## getAttributes

- Semantics: Calling this method will return the names and values of the editable attributes for the course named by the courseid parameter. These attributes can be updated with a call to updateAttributes, as detailed elsewhere in this doc.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for which the attributes will be retrieved

- Optional Arguments:
  - versionid : The specific version of the course to target for this call. If absent, use the most recent version.
- Example call: [web service instance]api?method=rustici.course.getAttributes&appid=myappid&courseid=course003(&versionid=2)
- Example response:

```

<attributes>
  <attribute name="alwaysFlowToFirstSco" value="false"/>
  <attribute name="commCommitFrequency" value="10000"/>
  <attribute name="commMaxFailedSubmissions" value="2"/>
  [...]
  <attribute name="validateInteractionResponses" value="true"/>
  <attribute name="wrapScoWindowWithApi" value="false"/>
</attributes>

```

- Error Codes:
  - 1 : Course named by courseid not found

## updateAttributes

- Semantics: This method allows you to update the attributes belonging to the course named by courseid. The attribute names and values are specified as name/value parameters in the HTTP request, see example usage below. Any faulty values given will be ignored. An error free call to this method will return a list of attributes that have been changed along with their new values. (If an attribute is set to it's existing value, it will not appear in the returned list. Only changed values will appear.)
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for which the attributes will be updated
- Optional Arguments:
  - versionid : The specific version of the course to target for this call. If absent, use the most recent version.
- Example call: [web service instance]api?method=rustici.course.updateAttributes&appid=myappid&courseid=course003&showNavBar=true&so on)
- Example response:

```

<attributes>
  <attribute name="showCourseStructure" value="false"/>
  <attribute name="showNavBar" value="true"/>
</attributes>

```

- Error codes:
  - 1 : Course named by courseid not found

## getCourseMetadata

- Semantics: A call to this method will return important metadata items associated with the specified course. Optional parameters can be included to specify both the scope of the metadata that is returned (course level only or activity level), as well as the format of the metadata (summary or detailed).
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for which the attributes will be updated

- Optional Arguments:
  - mdformat : The format of the return metadata. If specified as "course" return only data about the root learning object. If specified as "activity", return data for every activity / learning object in the course.
  - scope : If specified as summary, return only the most important subset of metadata such as title, description, duration, typical time, key words, and mastery score for each learning object. If specified as "full", return a much larger set of data. (\*Edit note from DE: The detailed report is currently unimplemented, I'm awaiting Troy's return to get to some of the code related to his metadata editor)
  - versionid : The specific version of the course to target for this call. If absent, use the most recent version.
- Example call: [web service instance]api?method=rustici.course.getCourseMetadata&appid=myappid&courseid=course003(&versionid=2&scope=summary)
- Example response:

- If mdformat is "course" or "activity": (\*Note only learning objects with metadata will contain a metadata element)

```
<object id="B0">
  <metadata>
    <title>Root Title</title>
    <description>Root Description</description>
    <duration>0</duration>
    <typicaltime>0</typicaltime>
    <keywords>
      <keyword>Training</keyword>
    </keywords>
  </metadata>
  [* if mdformat is "activity"
  <children>
    <object id="i1">
      <metadata>
        <title>Title</title>
        <description>Description</description>
        <duration>0</duration>
        <typicaltime>0</typicaltime>
        <keywords>
          <keyword>Training</keyword>
        </keywords>
        <masteryscore/>
      </metadata>
    </object>
  </children>]
</object>
```

- If mdformat is "full":

```
*Yet to be implemented, for now return same as summary
```

- Error codes:
  - 1 : Course named by courseid not found

## getCourseList

- Semantics: This method will return a list of courses associated with the given appid. If the optional filter parameter is specified, it will be used as a regular expression against the course id to filter the return list of courses.
- Required Arguments:
  - appid : Your application id
- Optional Arguments:
  - filter : A regular expression that will be used to filter the list of courses. Specifically only those courses whose courseid's match the given expression will be returned in the list.

- Example call: [web service instance]api?method=rustici.course.getCourseMetadata&appid=myappid(&filter=.\*321)
- Example response:

```
<courselist>
  <course id="test321" title="Photoshop Example -- Competency" versions="1" registrations="2"
 />
  <course id="course321" title="Another Test Course" versions="2" registrations="5" />
</courselist>
```

## Registration Service (rustici.registration)

This service provides calls for creating and removing registrations, and launching existing registrations.

### createRegistration

- Semantics: This method is used to create a new registration. To create new instances of existing registrations, createNewInstance should be used instead. A registration will contain a few pieces of information such as a learner name, a learner id, and optionally, information about where activity data should be posted (for client consumption), as well as a way to specify simple authentication schemes for posting said data, as noted below. A registration must be tied to a specific course at creation time. When the created registration is "launched", the course specified at creation time will be launched.
- Required Arguments:
  - appid : Your application id
  - courseid : The course for which this registration is being created
  - regid : The id used to identify this registration (it must be unique)
  - fname : The first name of the learner associated with this registration
  - lname : The last name of the learner associated with this registration
  - learnerid : The learner id associated with this registration (multiple registrations can be associated w/ the same learner id)
- Optional Arguments:
  - postbackurl : Specifies a URL for which to post activity and status data in real time as the course is completed
    - This status data is posted as 'data' (the key name) and starts with the <registrationreport ... > element and mirrors that schema found below.
  - authtype : Optional parameter to specify how to authorize against the given postbackurl, can be "form" or "httpbasic". If form authentication, the username and password for authentication are submitted as form fields "username" and "password", and the registration data as the form field "data". If httpbasic authentication is used, the username and password are placed in the standard Authorization HTTP header, and the registration data is the body of the message (sent as text/xml content type). This field is set to "form" by default.
  - urlname : You can optionally specify a login name to be used for credentials when posting to the URL specified in postbackurl
  - urlpass : If credentials for the postbackurl are provided, this must be included, it is the password to be used in authorizing the postback of data to the URL specified by postbackurl
  - resultsformat : This parameter allows you to specify a level of detail in the information that is posted back while the course is being taken. It may be one of three values: "course" (course summary), "activity" (activity summary, or "full" (full detail), and is set to "course" by default. The information will be posted as xml, and the format of that xml is specified below under the method "getRegistrationResult"

- versionid : Optionally, specify a particular version of the course specified by courseid above. If this parameter is missing, the most recent version of the package is used
- Example call: [web service instance]/api?method=rustici.registration.createRegistration&appid=myappid&courseid=course003&regid=myregurl]&authtype=form&urlname=iki&urlpass=secretish&resultformat=course&versionid=3)
  - \*Note that it could be fairly easy to exceed the maximum length of a query string based GET using this call, and hence the POST action should be used in those cases
- Example response:
 

```
<success/>
```
- Error Codes:
  - 1 : Couldn't find the course specified by courseid belonging to appid
  - 2 : Registration already exists
  - 3 : Postback URL login name specified without password

## createNewInstance

- Semantics: A call to this method will create a new instance of a registration, which is essentially just a copy of the registration that can be used without affecting the original. In some cases, a new instance of a registration will target the newest version of the course with which the original registration is associated. (\*Edit note: I think this behavior is ultimately set, right now, by the "Master" configuration file of the hosted engine instance... Should we figure out how to make that client specific, or just mention here what the expected behavior is (based on our setting)) The response for this method will indicate success and include the id of the newly created instance.
- Required Arguments:
  - appid : Your application id
  - regid : The identifier for the registration of which we want to create a new instance
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.registration.createNewInstance&appid=myappid&regid=myreg001
- Example response:
 

```
<success instanceid="1"/>
```
- Error codes:
  - 1 : The registration specified by regid does not exist

## deleteRegistration

- Semantics: A call to this method will attempt to delete the registration named by regid, and will consequently report success or throw an error.
- Required Arguments:
  - appid : Your application id
  - regid : The unique identifier for this registration
- Optional Arguments:
  - instanceid : This allows you to delete only a specific instance of this registration. By default all instances of the named registration will be deleted. This parameter can be specified as "latest" to delete only the latest instance of the registration.
- Example call: [web service instance]/api?method=rustici.registration.deleteRegistration&appid=myappid&regid=myreg001(&instanceid=2)
- Example response:

```
<success/>
```

- Error codes:
  - 1 : The registration specified by regid does not exist

## getRegistrationList

- Semantics: This method will return a list of registrations associated with the given appid. If the optional filter parameter is specified, it will be used as a regular expression against the registration id to filter the returned list of registrations. Also, another optional parameter, coursefilter, can be used as a regular expression against the course id to filter the list so that only registrations for courses matching the expression will be returned.
- Required Arguments:
  - appid : Your application id
- Optional Arguments:
  - filter : A regular expression that will be used to filter the list of registrations. Specifically only those registrations whose regid's match the given expression will be returned in the list.
  - coursefilter : A regular expression that will be used to filter the list of registrations. Specifically only those registrations that are associated with courses whose courseid's match the given expression will be returned in the list.
- Example call: [web service instance]api?method=rustici.course.getCourseMetadata&appid=myappid(&filter=.4&coursefilter=.321)
- Example response:

```
<registrationlist>
  <registration id="reg4" courseid="test321">
    <instances>
      <instance id="0" courseversion="0" />
      <instance id="1" courseversion="0" />
      <instance id="2" courseversion="1" />
      ...
    </instances>
  </registration>
  ...
</registrationlist>
```

## getRegistrationResult

- Semantics: A call to this method will return information about the specified registration, at a level of detail specified in the optional parameter "resultformat".
- Required Arguments:
  - appid : Your application id
  - regid : The unique identifier for this registration
- Optional Arguments:
  - resultformat : This optional parameter can be one of three values, "course", "activity", or "full". If specified as "course", only top level information such as completion status, success status, total time spent, and score will be returned. If specified as "activity", similar summary information will be return for every activity in the course. If specified as "full", the full set of CMI runtime and activity data is returned for every activity in the course. By default, "course" is used.
  - instanceid: This allows you to target a specific instance of the registration for this call. By default, use the latest instance.
- Example Call: [web service

instance]/api?method=rustici.registration.getRegistrationResult&appid=myappid&regid=myreg001(&instanceid=2

- Example Response:

- If resultsformat is "course" (or not specified)

```
<registrationreport format="summary" regid="myreg001" instanceid="0">
  <complete>complete</completion>
  <success>failed</success>
  <totaltime>19</totaltime>
  <score>0</score>
</registrationreport>
```

- If resultsformat is "activity"

```
<registrationreport format="activity" regid="myreg001" instanceid="0">
  <activity id="TOC1">
    <title>Photoshop Example -- Competency</title>
    <attempts>1</attempts>
    <complete>>false</complete>
    <success>>false</success>
    <time/>
    <score>0</score>
    <children>
      [if activity has children]
        <activity id="LESSON1">
          <title>Lesson 1</title>
          ...
          <score>0</score>
          <children>
            ...
          </children>
        </activity>
      <activity>
        ...
      </activity>
    </children>
  </activity>
</registrationreport>
```

- If resultformat is "full"

```
<registrationreport format="full" regid="myreg001" instanceid="0">
  <activity id="PRETEST">
    <title>Pre Assessment</title>
    <satisfied>>true</satisfied>
    <completed>>true</completed>
    <progressstatus>true</progressstatus>
    <attempts>1</attempts>
    <suspended>>false</suspended>
    <objectives>
      <objective id="PRIMARYOBJ">
        <measurestatus>>false</measurestatus>
        <normalizedmeasure>0.0</normalizedmeasure>
        <progressstatus>true</progressstatus>
        <satisfiedstatus>true</satisfiedstatus>
      </objective>
      ...
    </objectives>
    <runtime>
      <completion_status>completed</completion_status>
      <credit>Credit</credit>
      <entry>AbInitio</entry>
      <exit>Unknown</exit>
      <learnerpreference>
        <audio_level>1.0</audio_level>
        <language/>
        <delivery_speed>1.0</delivery_speed>
        <audio_captioning>0</audio_captioning>
      </learnerpreference>
      <location>null</location>
      <mode>Normal</mode>
      <progress_measure/>
      <score_scaled>68</score_scaled>
      <score_raw>534</score_raw>
```

```

<total_time>0000:00:00.00</total_time>
<timetracked>0000:00:04.47</timetracked>
<success_status>Unknown</success_status>
<suspend_data>user data]]&gt;&lt;/suspend_data&gt;
&lt;comments_from_learner&gt;
  &lt;comment&gt;
    &lt;value /&gt;
    &lt;location /&gt;
    &lt;date_time /&gt;
  &lt;/comment&gt;
  ...
&lt;/comments_from_learner&gt;
&lt;comments_from_lms&gt;
  &lt;comment&gt;
    ...
  &lt;/comment&gt;
  ...
&lt;/comments_from_lms&gt;
&lt;interactions&gt;
  &lt;interaction id="1"&gt;
    &lt;objectives&gt;
      &lt;objective id="1" /&gt;
      ...
    &lt;/objectives&gt;
    &lt;timestamp /&gt;
    &lt;correct_responses&gt;
      &lt;response id="1" /&gt;
      ...
    &lt;/correct_responses&gt;
    &lt;weighting /&gt;
    &lt;learner_response /&gt;
    &lt;result /&gt;
    &lt;latency /&gt;
    &lt;description /&gt;
  &lt;/interaction&gt;
  ...
&lt;/interactions&gt;
&lt;objectives&gt;
  &lt;objective id="PRIMARYOBJ"&gt;
    &lt;score_scaled/&gt;
    &lt;score_min/&gt;
    &lt;score_raw/&gt;
    &lt;score_max/&gt;
    &lt;success_status&gt;unknown&lt;/success_status&gt;
    &lt;completion_status&gt;unknown&lt;/completion_status&gt;
    &lt;progress_measure/&gt;
    &lt;description&gt;null&lt;/description&gt;
  &lt;/objective&gt;
  ...
&lt;/objectives&gt;
&lt;static&gt;
  &lt;completion_threshold/&gt;
  &lt;launch_data/&gt;
  &lt;learner_id&gt;daveid&lt;/learner_id&gt;
  &lt;learner_name&gt;dave e&lt;/learner_name&gt;
  &lt;max_time_allowed/&gt;
  &lt;scaled_passing_score/&gt;
  &lt;time_limit_action&gt;Undefined&lt;/time_limit_action&gt;
&lt;/static&gt;
&lt;/runtime&gt;
&lt;children&gt;
  [if activity has children]
  &lt;activity id="childactivity"&gt;
    ...
  &lt;/activity&gt;
  ...
&lt;/children&gt;
&lt;/activity&gt;
&lt;/registrationreport&gt;
</pre>
</div>
<div data-bbox="113 827 653 857" data-label="List-Group">
<ul style="list-style-type: none;">
<li>• Error codes:
      <ul style="list-style-type: none;">
<li>◦ 1 : The registration specified by the given regid does not exist</li>
</ul>
</li>
</ul>
</div>
<div data-bbox="94 883 326 898" data-label="Section-Header">
<h2>getRegistrationListResults</h2>
</div>
<div data-bbox="90 928 493 944" data-label="Page-Footer">
<p>Document generated by Confluence on Jun 05, 2009 14:56</p>
</div>
<div data-bbox="845 928 913 944" data-label="Page-Footer">
<p>Page 16</p>
</div>
```

- Semantics: This method is, as it is titled, a combination of `getRegistrationList` and `getRegistrationResult`, and can be used for very basic reporting functionality. The response contains the chosen list of registrations, and each registration includes a report of the same format received from a call to `getRegistrationResult`. The single report included for each registration is obtained from the most recent instance of the registration. (For report of another instance, use `getRegistrationResult` and specify the instance id.)
- Required Arguments:
  - `appid` : Your application id
- Optional Arguments:
  - `filter` : A regular expression that will be used to filter the list of registrations. Specifically only those registrations whose `regid`'s match the given expression will be returned in the list.
  - `coursefilter` : A regular express that will be used to filter the list of registrations. Specifically only those registrations that are associated with courses whose `courseid`'s match the given expression will be returned in the list.
  - `resultsformat` : This optional parameter can be one of three values, "course", "activity", or "full". If specified as "course", only top level information such as completion status, success status, total time spent, and score will be returned. If specified as "activity", similar summary information will be return for every activity in the course. If specified as "full", the full set of CMI runtime and activity data is returned for every activity in the course. By default, "course" is used.
- Example call: [web service instance]api?method=rustici.course.getCourseMetadata&appid=myappid(&filter=.4&coursefilter=.321&resultsf
- Example response:

```

<registrationlist>
  <registration id="reg4" courseid="test321">
    <instances>
      <instance id="0" courseversion="0" />
      <instance id="1" courseversion="0" />
      <instance id="2" courseversion="1" />
      ...
    </instances>
  </registration>
  ...
  [format of registrationreport element same as above in getRegistrationResult call]
  <registrationreport format="summary" regid="reg4" instanceid="2">
    ...
  </registrationreport>
</registrationlist>

```

## launch

- Semantics: Calling this method will launch the given registration by redirecting the user's browser to the main launch page for the course associated with the registration. Before redirecting the user, this method will also drop a cookie on the user's machine allowing them temporary access to the course materials. Unlike nearly all the other web service calls, this method is intended to be called directly in a browser.
- Required Arguments:
  - `appid` : Your application id
  - `regid` : The unique identifier for this registration
- Optional Arguments:
  - `instanceid` : Launch using a specific instance of the registration given, by default use the most recent instance
- Example call: [web service instance]/api?method=rustici.registration.launch&appid=myappid&regid=myreg001(&instanceid=2)
- Example response:

The user will be redirected to the launch page for the given registration.

- Error codes:
  - 1 : The registration specified by regid does not exist

## resetGlobalObjectives

- Semantics: Calling this method will reset all global objectives associated with this registration, if any exist.
- Required Arguments:
  - appid : Your application id
  - regid : The unique identifier for this registration
- Optional Arguments:
  - instanceid : Launch using a specific instance of the registration given, by default use the most recent instance
- Example call: [web service instance]/api?method=rustici.registration.resetGlobalObjectives&appid=myappid&regid=myreg001(&instanceid=)
- Example response:

```
<success/>
```

- Error codes:
  - 1 : The registration specified by regid does not exist

## FTP Access Service (rustici.ftp)

This service provides a highly configurable system for access to course files (both uploaded and unpacked/imported) via FTP. The intention of this FTP availability is largely geared to the case in which the user must upload entire courses or parts of courses that are much too large for a normal HTTP upload operation. Also, it exists to leverage the familiarity and convenience of FTP. For the service to provide convenient access while remaining secure and configurable, the interface for managing access is fairly extensive. However, many of the calls are unnecessary for very simple schemes of access. Regardless, two concepts must be understood in access management. The first of these is the concept of "permission domains". Permission domains are a way to isolate both uploaded files and courses into specific groups, such that any FTP user in a domain can only access an uploaded file or course belonging to the same domain. This allows an application which uses the web services to provide FTP access to several different clients, and to ensure that those clients do not have access to each other's files. The second concept in ftp access management is security tags. These allow more sophisticated access schemes, by providing a system of tagging both courses and users. The simple rule to remember when using these security tags is that any user who has been tagged with a certain tag (ex. 'TAGX') can see any course that has also been tagged with the same tag. (Note that the user, course, and tag must all belong to the same permission domain.) Because this extra level of configurability (and complexity) may only be needed in special cases, certain measures have been taken to ensure that they can be ignored by default. One result of this approach is that if a user has no tags, they will be able to access all courses in their permission domain. Therefore, if an application wishes to take advantage of security tags for FTP access, they must take caution to override this default behavior and tag a user with atleast one tag. Similarly, by default, all courses and users begin in the default permission domain, and must be explicitly moved to another domain if wished. The calls for these operations are listed below. In the simplest case, after a number of courses have been imported, an application can simply call createUser, and the user will have access to all courses in the default domain.

## createUser

- Semantics : A call to this method will create a new FTP user with the provided userid and password (userpass). Optionally, a permission domain can be specified as well, to place the user within that domain. Note that the permission domain must already exist (and can be created using the createPermissionDomain call listed below).
- Required Arguments:
  - appid : Your application id
  - userid : The id of the user. This id will become the user's login name when connecting to FTP.
  - userpass : The password for the user, to be used in authentication when connecting to FTP.
- Optional Arguments:
  - pd : The permission domain in which to place this new user. Note this permission domain must already exist.
- Example call: [web service instance]/api?method=rustici.ftp.createUser&appid=myappid&userid=testuser&userpass=ftpsecret(&pd=testdom
- Example response:

```
<success />
```
- Error codes:
  - 1 : A user with the given userid already exists
  - 2 : The optional permission domain specified does not exist

## setUserPassword

- Semantics: A call to this web method will update the password for the user specified by the given userid. The old password must be given in the call, and will be used for authentication in changing the password to the new value.
- Required Arguments:
  - appid : Your application id
  - userid : The id of the user owning the password to be changed.
  - oldpass : The existing password of the user.
  - newpass : The new password for the given user.
- Example call: [web service instance]/api?method=rustici.ftp.setUserPassword&appid=myappid&userid=testuser&oldpass=ftpsecret&newpass
- Example response:

```
<success />
```
- Error codes:
  - 1 : The user with the given userid does not exist
  - 2 : The oldpass parameter did not match the existing password for this user

## deleteUser

- Semantics : This method deletes an existing FTP user, specified by the userid parameter.
- Required Arguments:
  - appid : Your application id
  - userid : The id of the user to delete.
- Optional Arguments: None
- Example call: [web service

instance]/api?method=rustici.ftp.deleteUser&appid=myappid&userid=testuser

- Example response:

```
<success />
```

- Error codes:
  - 1 : The user with the given userid doesn't exist

### createPermissionDomain

- Semantics : A call to this method creates a new permission domain (in which users, courses, and tags can be placed via other calls.)
- Required Arguments:
  - appid : Your application id
  - pd : The name of the new permission domain to be created
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.createPermissionDomain&appid=myappid&pd=testdomain
- Example response:

```
<success />
```

- Error codes:
  - 1 : A permission domain with the given name already exists
  - 2 : There was a file system error associated with creating the permission domain

### deletePermissionDomain

- Semantics: This method deletes an existing permission domain
- Required Arguments:
  - appid : Your application id
  - pd : The name of the existing permission domain to be deleted
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.deletePermissionDomain&appid=myappid&pd=testdomain
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : There was a file system error associated with deleting the permission domain

### getDomainList

- Semantics: This method returns a list of existing permission domains, which can be used without error in various other service calls.
- Required Arguments:
  - appid : Your application id
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.getDomainList&appid=myappid

- Example response:

```
<domains>
  <domain id="testdomain"/>
  <domain id="domain1"/>
  ...
</domains>
```

- Error codes: No call-specific codes

## setCourseDomain

- Semantics: This method will place the course specified by the given courseid into the given domain. Both the course and domain must already exist. After placing a course in a given permission domain, only users that are also placed in the same domain will be able to access the course through FTP.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course to be placed in this permission domain
  - pd : The permission domain in which to place the given course
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.setCourseDomain&appid=myappid&courseid=course003&pd=testdomain
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified course does not exist

## setUserDomain

- Semantics: This method will place the user specified by the given userid into the given domain. Both the user and domain must already exist. After placing a user in a given permission domain, the user will be able to see (only) the courses that have also been placed in the same domain.
- Required Arguments:
  - appid : Your application id
  - userid : The id used to identify the user to be placed in this permission domain
  - pd : The permission domain in which to place the given user
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.setUserDomain&appid=myappid&userid=testuser&pd=testdomain
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified user does not exist

## createTag

- Semantics: This method will create a new tag under the specified permission domain. This tag can then be used in subsequent tagCourse and tagUser calls.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain in which to create the tag
  - tag : The name of the tag
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.createTag&appid=myappid&pd=testdomain&tag=testtag
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag already exists in this permission domain

### deleteTag

- Semantics: This method will delete an existing tag from the specified permission domain.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain in which to create the tag
  - tag : The name of the tag
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.deleteTag&appid=myappid&pd=testdomain&tag=testtag
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag does not exist in this permission domain

### tagCourse

- Semantics: This method will tag the given course with the given security tag. Only users which belong in the same permission domain as the course and have been tagged with the same security tag will be able to access the course upon FTP login.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain under which the tag and course belong
  - tag : The name of the tag
  - courseid : The id used to identify the course that will be tagged
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.tagCourse&appid=myappid&pd=testdomain&tag=testtag&courseid=course003
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag does not exist in this permission domain
  - 3 : The specified course cannot be found

## untagCourse

- Semantics: This method will remove the security tag from the given course.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain under which the tag and course belong
  - tag : The name of the tag
  - courseid : The id used to identify the course from which the tag will be removed
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.untagCourse&appid=myappid&pd=testdomain&tag=testtag&courseid=course00
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag does not exist in this permission domain
  - 3 : The specified course cannot be found

## tagUser

- Semantics: This method will tag the given user with the given security tag. Only courses which belong in the same permission domain as the user and have been tagged with the same security tag will be able available to the user upon FTP login.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain under which the tag and user belong
  - tag : The name of the tag
  - userid : The id used to identify the user that will be tagged
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.tagUser&appid=myappid&pd=testdomain&tag=testtag&userid=testuser
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag does not exist in this permission domain
  - 3 : The specified user cannot be found

## untagUser

- Semantics: This method will remove the security tag from the given user.
- Required Arguments:

- appid : Your application id
- pd : The permission domain under which the tag and user belong
- tag : The name of the tag
- userid : The id used to identify the user from which the tag will be removed
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.untagUser&appid=myappid&pd=testdomain&tag=testtag&userid=testuser
- Example response:

```
<success />
```

- Error codes:
  - 1 : The specified permission domain does not exist
  - 2 : The specified tag does not exist in this permission domain
  - 3 : The specified user cannot be found

### getDomainInfo

- Semantics: This method will return information about the given permission domain. Specifically, it will return information about the tags, users, and courses that have been assigned to the given domain.
- Required Arguments:
  - appid : Your application id
  - pd : The permission domain for which to retrieve info
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.getDomainInfo&appid=myappid&pd=testdomain
- Example response:

```
<domain id="testdomain">
  <tags>
    <tag id="testtag"/>
    ...
  </tags>
  <users>
    <user id="testuser"/>
    ...
  </users>
  <courses>
    <course id="test321"/>
    ...
  </courses>
</domain>
```

- Error codes:
  - 1 : The specified permission domain does not exist

### getCourseInfo

- Semantics: This method will return information about the tags and domain associated with the course specified by the given courseid.
- Required Arguments:
  - appid : Your application id
  - courseid : The id used to identify the course for which the information will be retrieved
- Optional Arguments: None
- Example call: [web service

instance]/api?method=rustici.ftp.getCourseInfo&appid=myappid&courseid=test321

- Example response:

```
<course id="test321" domain="domain1">
  <tags>
    <tag id="tag1"/>
    ...
  </tags>
</course>
```

- Error codes:
  - 1 : The specified course could not be found

## getUserInfo

- Semantics: This method will return information about the tags and domain associated with the user specified by the given userid.
- RequiredArguments:
  - appid : Your application id
  - userid : The id used to identify the user for which the information will be retrieved
- Optional Arguments: None
- Example call: [web service instance]/api?method=rustici.ftp.getUserInfo&appid=myappid&userid=testuser
- Example response:

```
<ftpuser id="testuser" domain="domain1">
  <tags>
    <tag id="tag1"/>
    ...
  </tags>
</ftpuser>
```

- Error codes:
  - 1 : The specified user could not be found