**Engineering Competition REST Service**

Copyright 2017, Clinton Staley

## Updates

**6/1/21 Revised NOT\_FOUND ambiguity: removed NOT\_FOUND from Error Codes section.**

**6/18/21 Changed Ssn identifier from cookie to id**

**Overview**

The Virtual Design Competition Service (VCS) provides a REST interface for managing virtual science and engineering design competitions. It tracks students, teachers, classes, and competitions.

**General Points**

The following design points apply across the document.

1. All resource URLs are prefixed by some root URL, (e.g. http://www.softwareinventions.com/VCS/)
2. All resources accept and provide only JSON body content. And per REST standards, all successful (200 code) DELETE actions return empty body.
3. Some GET operations allow get-parameters. These are listed directly after the GET word. All get-parameters are optional unless given in bold.
4. Absent documentation to the contrary, all DELETE calls, POST, and PUT calls with a non-200 HTTP response return as their body content, a list of JSON objects describing any errors that occured. Error objects are of form {tag: {errorTag}, params: {params}} where errorTag is a string tag identifying the error, and params is a possibly-empty array of additional values needed to fill in details about the error. E.g. {tag: "missingField", params: ["lastName"]}
5. Resource documentation lists possible errors only when the error is not obvious from this General Points section. Relevant errors may appear in any order in the body. Missing field errors are checked first, and no further errors are reported if missing fields are found.
6. All resource-creating POST calls return the newly created resource as a URI via the Location response header, not in the response body. The respose body for such POSTs is reserved for error information, per point 4.
7. GET calls return one of the following, returning the earliest one that is applicable. Response body is empty in cases b-d.
   1. 500 for any server error, as described under points 11, 12, and 15
   2. UNAUTHORIZED for missing login.
   3. FORBIDDEN for insufficient authorization despite login
   4. NOT\_FOUND for a URI that is not described in the REST spec if logged in, 401 if not.
   5. BAD\_REQUEST and a list of error objects for any errors checkable without any database consultations.
   6. HTTP code OK and the specified information in the body.
8. GET calls whose specified information is a list always return an array, even if it has just one or even zero elements.
9. Fields of JSON content for POST and PUT calls are assumed to be strings, booleans, ints, or doubles without further documentation where obvious by their name or intent. In nonobvious cases, the docs give the type explicitly.
10. All access requires authentication via login to establish the Authenticated User (AU); no resources are public except for Prss/POST (for initial registration), and Ssns/POST (to log in). Other resources may be restricted based on admin status of AU. The default restriction is to allow only access relevant to the AU, unless the AU is admin, in which case access to any Person's info is allowed.
11. Any database query failure constitutes a server error (status 500) with a body giving the error object returned from the query. Ideally, no request, however badly framed, should result in such an error except as described in point 12.
12. The REST interface does no general checking for forbiddenField errors, unless the spec specifically indicates it will. Absent such checking, non-specified body fields in PUT/POST calls may result in database query errors and a 500 code, as may an empty body when body content is expected.
13. All body fields in POST calls are required unless expressly specified as optional. By contrast, all fields in PUT calls are optional unless expressly specified as required. Required fields may not be passed as null, undefined or "". Doing so has the same outcome as if the field were entirely missing
14. All times are integer values, in mS since epoch.
15. Non JSON parseable bodies result in 500 error.

**Error Codes**

The possible error codes, and any parameters, are as follows.

*missingField* Field missing from request. Params[0] gives field name

*badValue* Field has bad value. Params[0] gives field name

*badLogin* Email/password combination invalid, for errors logging.

*dupEmail* Email duplicates an existing email

*noTerms* Acceptance of terms is required

*noOldPwd* Change of password requires an old password

*oldPwdMismatch* Old password that was provided is incorrect.

*dupTitle* Competition title duplicates an existing one

*dupEnrollment* Duplicate enrollment

*forbiddenField* Field in body not allowed. Params[0] gives field name.

*queryFailed* Query failed (server problem)

*cantRemoveLeader* Attempted Removal of Team Leader

*noCompType* can't find the competition type supplied to a competition, used when creating a competition

*invalidPrms* prms supplied to a competition don't pass the prmSchema of the competition type

*badTeamLead* new team leader specified that isnt in the current team

**Resources for User Management, including Registration**

**(Admin use in purple)**

**Prss**

Collection of all current students or other users.

***GET*** email={email or email prefix}

Returns list of zero or more Persons. Limits response to Persons with specified email or email prefix, if applicable. Data per person:

*email* principal string identifier, unique across all Persons

*id* id of person with said email, so that URI would be Prss/{id}

***POST***

Adds a new Person. No AU required, as this resource/verb is used for registration, but an AU is allowed, and an admin AU gets special treatment as indicated.

*email* unique Email for new person

*firstName*

*lastName*

*password*

*role* 0 for student, 1 for admin

*termsAccepted* boolean--were site terms and conditions accepted?

Error if email is nonunique. Error if terms were not accepted and AU is not admin. Error forbiddenRole if role is not student unless AU is admin.

**Prss/{prsId}**

***GET***

Return a single Person {prsId}, with fields as specified in POST for Prss, plus dates *termsAccepted* and *whenRegistered*, less *password*. (*termsAccepted* may be falsey if terms were not accepted.) The dates give time of term acceptance and registration, and will generally be equal, but are listed separately for legal reasons.

***PUT***

Update Person {prsId}, with body giving an object with one or more of *firstName*, *lastName*, *password*, *role.* Attempt to change other fields in Person such as *termsAccepted* or *whenRegistered* results in BAD\_REQUEST and forbiddenField error(s). Role changes result in BAD\_REQUEST with badValue tag for nonadmins. All changes require the AU be the Person in question, or an admin. Unless AU is admin, an additional field *oldPassword* is required for changing *password*. Password, if supplied, must be nonempty.

***DELETE***

Delete the Person in question, including all Cmps owned by Person. Requires admin AU.

**Prss/{prsId}/Cmps**

***GET***

Return an array with one element for each Cmp the Prss is subscribed to, with fields for each as given. If a Prs is on two or more teams for the same Cmp, return only one instance of the Cmp.

*id* Cmp Id

*ownerId* Prs Id of owner

t*itle* Cmp title

*prms* Cmp parameters

*rules*

*curTeamId* Id of team currently making a submit if round-robin

**Prss/{prsId}/Teams**

***GET***

Return array with one element for each Team the Prss is a member of, in form Cmps/{cmpId}/Teams/{teamId}. Fields are:

*id* Id of the Team

*teamName* Name of the Team, must be unique per Cmp

*leaderId* Id of the team Leader

*bestScore* Best score thus far earned by submissions from the Team. 0 if no submits

*lastSubmit* Date of last submission from Team. Null if no submits.

*canSubmit* boolean indicating if submission is allowed. If the Cmp's submitRules are standard, all teams can submit at any time. If submitRules are round-robin, on the team next in order after the most recent submit may do so.

*cmpId* Id of the competition on which the team is working

*nextTeam* Id of the next team in round-robin order or null if N/A

**Ssns**

Login sessions (Ssns) establish an AU and are required for most service access. A user obtains one via POST to Ssns.

***GET***

Return a list of all active sessions. Admin-privileged AU required. Return array of

*cookie* Unique cookie value for session

*prsId* ID of Person logged in

*loginTime* Date and time of login

***POST***

A successful POST generates a browser-session cookie that will permit continued access for 2 hours. Indicated Person becomes the AU. An unsuccesful POST results in a 400 with a badLogin tag and no further information.

*email* Email of user requesting login

*password* Password of user

**Ssns/{id}**

***GET***

Returns, for the indicated session, a single object with same properties as one element of the array returned from Ssns GET. AU must be admin or owner of session.

***DELETE***

Log out the specified Session. AU must be owner of Session or admin.

**Resources for Competition Types**

The following resources allow creation, deletion, and management of Competition Types (Ctps). A competition type is a kind of design competition, e.g. a model bridge building competition or a catapult design competition. It includes a general description of the competition for the participant's benefit, and a JSON Schema describing allowed parameters (see Cmp below). For all POST, PUT, or DELETE operations, AU must be admin.

**Ctps**

***GET***

Any AU is acceptable, though some login is required. Return an array of 0 or more elements, with one element for each Ctp in the system.

*id* Id of the Competition Type

*title* Title of the Competition Type

*description Human readable*description of the Competition Type, in MD format.

*tutorial* tells user the rules of the game and how to play

*prmSchema* JSON schema to which parameters must conform. May be null. Null implies no parameters allowed for this Ctp

***POST***

Create a new Ctp. Error dupTitle if title is a duplicate. Fields are

*title* Title of the new Competition, limited to 80 chars

*codeName* One word name for the competition, for internal identification

*description* General description of the Competition Type, in MD format.

*tutorial* tells user the rules of the game and how to play.

*prmSchema* the schema for this Competition Type

**Ctps/{CtpId}**

***GET***

Return single object having same properties as one of the array elements returned by Ctps GET, for just the indicated Ctp. Any AU is acceptable though some login is required.

***PUT***

Update the title, description or prmSchema of the Ctp Error dupTitle if title is duplicate.

***DELETE***

Delete the Ctp, including all associated Cmps

**Resources for Competitions**

The following resources allow creation, deletion, and management of Competitions (Cmps). These are individual instances of a Ctp, e.g. a particular bridge-building competition. Each Ctp has an owner, e.g. the sponsoring instructor, possibly some specialized parameters (prms), e.g. a list of allowed parts, some ground rules, e.g. whether other teams' work is visible, and a set of participating teams.

**Cmps**

***GET*** owner=<email>, CompetitionType = <id>

Any AU is acceptable, though some login is required. Return an array of 0 or more elements, with one element for each Competition in the system, limited to Competitions with the specified owner if query param is given:

*id* Id of the Competition

*ownerId* Owner of the Competition

*ctpId* Id of the relevant Ctp

*title* Title of the Cmp, unique per owner.*rules* 0 for standard, 1 for round-robin

*prms* Parameters for this Cmp, in JSON form.

*description* description specific to this competition, offering e.g. more hints and detail than the general Ctp description.*rules* 0 for standard, 1 for round-robin

***POST***

Create a new Competition, owned by the current AU. Error dupTitle if title is a duplicate. Error noCompType if invalid Competition Type is specified. invalidPrms error if prms isn't valid according to prmSchema of Competition Type. Fields are:

*ctpId* Id of the Competition Type

*title* Title of the new Competition, limited to 80 chars

*prms* Parameters for this Cmp, in JSON form. Not null, though may be empty object, e.g. {}

*rules* 0 for standard, 1 for round-robin

*description* description specific for a cmp type

**Cmps/{CmpId}**

***GET***

Return single object having same properties as one of the array elements returned by Cmps GET, for just the indicated Cmp Any AU is acceptable though some login is required.

***PUT***

Fields as for Competitions POST. Error dupTitle if title is duplicate. invalidPrms error if prms isn't valid according to prmSchema of Competition Type. AU must be Competition owner or admin.

***DELETE***

Delete the Competition, including all associated Teams Submissions. AU must be Competition owner or admin.

**Cmps/{CmpId}/WaitingSbms**

***GET num=<n>***

Behavior similar to that of Cmps/{CmpId}/Teams/{teamId}/Sbms GET, but with some additional fields, numSubmits, lastSubmit and bestScore. Drawing from the pool of Sbms awaiting EVC attention (those with null Responses), and requiring admin AU.

**Cmps/{CmpId}/Teams**

Resources to track Teams participating in a given Cmp. For all verbs in this section, unless otherwise specified, any AU is acceptable, though some login is required. This implies general visibility and openness of all competitions, which may change in the future but is simpler for now.

***GET***

Return an array of 0 or more elements, with one element for each Team in the Cmp.

*id* Id of the Team

*teamName* Name of the Team, must be unique per Cmp

*leaderId* Id of the team Leader

*cmpId Id of the competition*

*bestScore* Best score thus far earned by submissions from the Team. 0 if no submits

*lastSubmit* Date of last submission from Team. Null if no submits.

*numSubmits Number of submissions thus far made. (This is denormalized as submission count could be obtained by fetching all submissions, but this value is needed for some grading models and is best made available here.)*

*canSubmit* boolean indicating if submission is allowed. If the Cmp's submitRules are standard, all teams can submit at any time. If submitRules are round-robin, on the team next in order after the most recent submit may do so.

***POST***

Create a new Team, owned by the current AU, who becomes the *team lead*. Error dupTitle if title is a duplicate. Fields are

leaderId Id of the leader. Does not have to be AU

*teamName* Name of the Team, must be unique per Cmp

**Cmps/{CmpId}/Teams/{TeamId}**

***GET***

Return single object having same properties as one of the array elements returned by ../Teams GET, for just the indicated Team.

***PUT***

Update the title of the Team and/or change leader. Error dupTitle if title is duplicate within the same competition. Error badTeamLead if leaderId is not presently a team member. AU must be Team lead, Cmp owner, or admin. Fields, all optional:

name New team name (may duplicate original w/o error)

leaderId Person ID of team leader

***DELETE***

Delete the Team, including all associated Submissions. AU must be the team leader or an admin.

**Cmps/{CmpId}/Teams/{TeamId}/Mmbs**

***GET***

Return an array of all team members each element of which has the following fields

*id* id of the member

*firstName* first name of the member

*lastName* last name of member

*email* email of member

***POST***

Add a member to the team. To add must be the team leader or admin, Fields are:

*prsId* Id of the new member

**Cmps/{CmpId}/Teams/{TeamId}/Mmbs/{prsId}**

***DELETE***

Delete the member, AU must be team leader, admin, or the member being removed. The team leader cannot be removed from the team by any AU, the only way for the leader to leave the team is to first assign another leader, then delete the original team leader.

**Resources for Submissions**

Teams make submissions to the Competition -- attempts to solve the design challenge.

**Cmps/{CmpId}/Teams/{TeamId}/Sbms**

***GET num=<n>***

Return an array of 0 or more elements, each giving information on a submission by the relevant Team to the relevant Cmp, in reverse chronological order. If num is provided, return at most that many Sbms.

*id* Id of the submit

*cmpId* Id of competition submission is in

*teamId* Id of team making submit

*time* Date and time of the submit

*score* Score given the submit -- double

*content* JSON-format string, with content format specific to the Ctp in question . This is what the team submitted as their answer

*response* JSON-format string detailing the system's response to the submission. This data may be used by clientside app to e.g. show the result of the submit in diagram or even animated form.

*practiceRun* boolean telling grader to grade fully or not

***POST***

Make a new submit on behalf of this Team for the relevant Cmp. AU must be a team member or the Cmp owner if only content is posted. AU must be admin if response is posted. Fields, each optional, are as defined in GET This POST also sets cansubmit to false, and updates numSubmits and lastSubmit, if relevant.

*content* JSON-format string, with content format specific to the Ctp in question.

**Cmps/{CmpId}/Teams/{TeamId}/Sbms/{SbmId}**

***GET***

Return single object having same properties as one of the array elements returned by ../Sbms GET, for just the indicated Sbm.

***PUT***

Update Sbm in question by adding a response. Requires admin AU.

*testResult* JSON-format string, with content format specific to the relevant Ctp

*score* score as an double given by the grader, updates bestScore and lastScore appropriately

errorResult text error string if necessary

canSubmit will change the team canSubmit property

**Special DB Resource for Testing Purposes**

**DB**

***DELETE***

Clear all content from the database, reset all autoincrement IDs to 1, and add back one Person, an admin named Joe Admin with email adm@11.com and password "password". Clear all current sessions. AU must be an admin.