ChatGPT

Canvas LMS REST API – Guide to Push Operations (Create/Update Only)

This guide covers **Canvas LMS REST API endpoints** (for an Instructure-hosted Canvas instance) that are used **exclusively for creating or updating data** – i.e. "push" operations. Each section is organized by resource (Courses, Assignments, Groups, etc.), and for each endpoint we list the HTTP method, path, and a description of its function. Required parameters are noted, and key optional parameters are mentioned (for full details, see the official Canvas API documentation).

Note: All endpoints are under the base URL `https://<canvas-domain>/api/v1/`. We do not include any GET (read/list) or DELETE operations, only POST/PUT that create or modify resources.

Courses

- **Create a Course** (`POST /api/v1/accounts/:account_id/courses`): Creates a new course under the specified account. **Required:** the `account_id` in the path and usually a course name (if not provided, it defaults to "Unnamed Course" [1]). Supports many optional fields to set course properties, such as `course[course_code]` (course code), start and end dates (`start_at`, `end_at` – used if `restrict_enrollments_to_course_dates` is true), license, enrollment settings (e.g. `is_public`, `open_enrollment`), etc [2] [3].
- **Update a Course** (`PUT /api/v1/courses/:id`): Updates an existing course's settings. Accepts the same fields as course creation (name, course_code, dates, etc.) to modify the course's attributes [4] [5]. For example, you can change the course name or course code, adjust start/end dates, or update visibility settings. (If a user only has content editing rights, they may only update certain fields like the syllabus body [6].) This endpoint requires the course ID in the path.

Assignment Groups (Assignment Categories)

- **Create an Assignment Group** (`POST /api/v1/courses/:course_id/assignment_groups`): Creates a new assignment group (category) in the given course. **Required:** `name` – the name of the assignment group [7]. Optional parameters include `position` (numeric order of the group in the course's list) and `group_weight` (the percentage of the total grade this group counts for, if using weighted grading) [8]. You can also set `sis_source_id` for SIS integration and provide grading policy `rules` if needed [9] [10].
- **Update an Assignment Group** (`PUT /api/v1/courses/:course_id/assignment_groups/:assignment_group_id`): Modifies an existing assignment group's properties. Accepts the same fields as creation (name, position, group_weight, etc.) to update the category [11] [12]. For example, you can rename the assignment group or change its weight. The `assignment_group_id` is specified in the path.

Assignments

- **Create an Assignment** (`POST /api/v1/courses/:course_id/assignments`): Creates a new assignment in a course. **Required:** `assignment[name]` – the assignment's title [13]. (If no name is provided, Canvas will not create the assignment.) You can specify many optional fields to configure the assignment, such as: `assignment[submission_types]` (one or multiple submission methods: online, on-paper, quiz, etc.) [14] [15], `assignment[allowed_extensions]` (file types if file upload is allowed) [16] [17], `assignment[points_possible]`, due date (`assignment[due_at]`), availability dates (`unlock_at` / `lock_at`), whether it's a group assignment (`group_category_id`), peer review settings, and more. The assignment is created in the "active" (published) state by default [18].
- **Update an Assignment** (`PUT /api/v1/courses/:course_id/assignments/:id`): Updates an existing assignment's settings. Accepts the same fields as assignment creation to change the assignment's properties (title, description, points, dates, submission settings, etc.) [19] [20]. For example, you can adjust the due date or change the submission type. Note that some fields cannot be changed after students have submitted (for instance, you cannot switch the `submission_types` if there are submissions). The assignment ID to update is in the path.

Group Categories (Group Sets)

- **Create a Group Category** (`POST /api/v1/courses/:course_id/group_categories`): Creates a new group category (also called a *group set* in the UI) in a course. **Required:** `name` – the name of the group category [21]. Optional settings include: `self_signup` (allow students to self-enroll in groups; can be `"enabled"`, `"restricted"` for same-section only, or blank for no self-signup) [22] [23], `auto_leader` (automatically assign group leaders, e.g. `"first"` member or `"random"` member) [24] [25], and `group_limit` (maximum members per group, if self-signup is enabled) [26]. You may also specify `create_group_count` to pre-create a number of empty groups in this category [27].
- **Update a Group Category** (`PUT /api/v1/group_categories/:group_category_id`): Modifies an existing group category's settings. You can update the `name` of the group category, toggle `self_signup` on or off (or change its mode), and adjust other settings like `group_limit` [28] [29]. (Any field not provided in the PUT request will remain unchanged.) The group category ID is in the path (note: this endpoint is not scoped by course ID in the URL).

Groups

- **Create a Group** (`POST /api/v1/group_categories/:group_category_id/groups`): Creates a new group within a given group category (course group set). **Required:** typically a group `name` (if not provided, it may default to something like "Unnamed Group") [30]. You can also provide a `description` for the group. If you use the alternative endpoint `POST /api/v1/groups` (without specifying a category), the group will be created as a "community" group not tied to a course [31] [32]. Optional parameters for group creation include `is_public` (if using community groups, this toggles whether the group is visible to the public) and `join_level` – which controls how members join: `"invitation_only"` (default), `"parent_context_auto_join"` (open enrollment within the course), or `"parent_context_request"` (request to join) [33] [34].

- **Update a Group** (`PUT /api/v1/groups/:group_id`): Updates an existing group's properties. You can change the group's `name` or `description`, and (for community groups) you can set `is_public` (note: once a group is made public, it cannot be set back to private) [35] [36] . You may also adjust the `join_level` (e.g., to switch between invite-only and self-join) [34] . Additionally, this endpoint allows setting a group avatar image by using the `avatar_id` of an uploaded file [37] , and you can manage membership through the `members[]` parameter (invite or remove specific users in one call) [38] .

Students (Enrollments and Group Memberships)

- **Enroll a User in a Course** (`POST /api/v1/courses/:course_id/enrollments`): Adds a user to a course with a given enrollment type (role). **Required:** `enrollment[user_id]` – the Canvas user ID (or SIS ID with format `sis_user_id:XXX`) of the person to enroll, and `enrollment[type]` – the role type for enrollment [39] [40] . For example, use `"StudentEnrollment"` to add a student, or `"TeacherEnrollment"` for a teacher. Optional fields include `enrollment[course_section_id]` (to enroll into a specific section of the course) [41] , and `enrollment[enrollment_state]` to control activation: set to `"active"` to enroll immediately, or default `"invited"` to require the user to accept an invitation [42] . (You can also set `"inactive"` if you want to add the student in an inactive state.) Other options: `enrollment[notify]` (if true, sends a notification to the user about the enrollment) [43] .
- **Add User to a Group** (`PUT /api/v1/groups/:group_id/users/:user_id`): Adds a user to a group within a course (or accepts their membership request). The group ID and the user ID are in the URL. This operation will create a group membership with status "accepted" (active member) [44] . There are no required body parameters apart from the path IDs – performing this PUT on a user who isn't already in the group will add them. You can optionally include `moderator=true` in the body if you want to grant the user moderator rights in the group (i.e. make them a group leader) [45] [46] . (To remove a user from a group, a DELETE endpoint is used instead, which is outside the scope of this guide.)

Rubrics

- **Create a Rubric** (`POST /api/v1/courses/:course_id/rubrics`): Creates a new grading rubric in the course. **Required:** a rubric title and at least one criterion definition (the API expects a complex `rubric[criteria]` hash if you supply custom criteria; otherwise an automatic criterion can be generated). In practice, you must provide `rubric[title]` (e.g. "My Rubric") [47] and criteria details. This endpoint is a bit special – it returns a combined result containing both a `Rubric` object and a `RubricAssociation` object [48] [49] . Optional parameters include `rubric[free_form_criterion_comments]` (allow free-form comments) and you can immediately create an association by specifying `rubric_association[association_id]` (e.g. an assignment ID) and `association_type` (`"Assignment"` , `"Course"` , etc.) [50] [51] . You can also set `use_for_grading` in the association to true if this rubric should be used for assignment grading [52] .
- **Update a Rubric** (`PUT /api/v1/courses/:course_id/rubrics/:id`): Updates an existing rubric's definition. You can change the `rubric[title]` , modify criteria or rating descriptions, etc. This uses the same fields format as creation (you would provide a revised set of rubric criteria, title, or association settings in the PUT). Like the create endpoint, the response returns a rubric +

association composite [53] [54] . (For example, you might rename the rubric or adjust point values via this endpoint.)

- **Attach Rubric to Assignment** (`POST /api/v1/courses/:course_id/rubric_associations`): Creates a Rubric Association, which links a rubric to an assignable object (such as an assignment) in the course. **Required:** `rubric_association[rubric_id]` (the ID of an existing rubric), `rubric_association[association_id]` (the ID of the object to attach the rubric to, e.g. an assignment ID), and `rubric_association[association_type]` (the type of object, e.g. `"Assignment"` for an assignment) [55] [56] . Optional fields: `rubric_association[use_for_grading]` (boolean, set true if the rubric is used to grade the assignment) [57] , `rubric_association[hide_score_total]` (if true, hide the rubric's total score from students) [58] , and `rubric_association[purpose]` (`"grading"` vs `"bookmark"` – use `"grading"` for attaching to an assignment) [59] . This call returns the created RubricAssociation object linking the rubric to the assignment [60] [61] . (To update an existing association, use `PUT /api/v1/courses/:course_id/rubric_associations/:id` with similar fields [62] [63] .)

Pages (Course Wiki Pages)

- **Create a Page** (`POST /api/v1/courses/:course_id/pages`): Creates a new Wiki page in a course's Pages section. **Required:** `wiki_page[title]` – the title of the page [64] . Optional fields include `wiki_page[body]` for the page content (HTML), `wiki_page[published]` to set the publish state (`false` will save as a draft) [65] , and `wiki_page[editing_roles]` to restrict who can edit the page (e.g. `"teachers"`, `"students"`, etc., comma-separated) [66] [67] . You can also set `wiki_page[notify_of_update]` (whether to notify course members of changes) [68] . The response will include the page's `url` (unique identifier based on the title).
- **Update a Page** (`PUT /api/v1/courses/:course_id/pages/:url`): Updates an existing course page's content or settings. You identify the page by its `:url` (or page ID using the `page_id:<id>` format if needed). You can change the `wiki_page[title]` (if you do so, the page's URL will change to match the new title) [69] [70] , as well as the `body`, `editing_roles`, `published` flag, etc., just like in page creation. For example, you might use this to edit the page's text or to publish/unpublish a page. **Note:** To update the special Front Page of a course, the endpoint is `PUT /courses/:course_id/front_page` (with the same parameters) [71] .

Discussion Topics

- **Create a Discussion** (`POST /api/v1/courses/:course_id/discussion_topics`): Creates a new discussion topic in a course. **Required:** `title` – the discussion title [72] . Optional fields allow extensive customization: `message` (the discussion's text prompt, in HTML) [73] , `discussion_type` (defaults to a focused discussion if not set; allowed values: `"side_comment"` or `"not_threaded"` for focused style, or `"threaded"` for fully threaded discussions [74]), `published` (true to publish immediately or false to create as a draft) [75] , and scheduling fields like `delayed_post_at` (to schedule a future publish date) [76] and `lock_at` (date to lock the discussion). You can also enable podcast feeds (`podcast_enabled`), require users to post before seeing replies (`require_initial_post`) [77] , and if creating a graded discussion, include an `assignment` sub-object with assignment settings (points, due date, etc.) [78] . Setting `is_announcement=true` will create an Announcement instead of a Discussion (only users with permission can do this) [79] .

- **Update a Discussion** (`PUT /api/v1/courses/:course_id/discussion_topics/:topic_id`): Updates an existing discussion topic (identified by its ID in the path). You can modify the same fields that are used in creation. For example, you may change the `title` or `message` (description) of the discussion [80], adjust `discussion_type` (focused vs threaded) [81], or toggle `published` / draft status [82]. Timing fields like `delayed_post_at` and `lock_at` can be updated to reschedule or enforce locks [83]. You can also pin/unpin the discussion (`pinned` flag) or even convert it to an announcement by setting `is_announcement` if appropriate [84]. Essentially, any attribute from the creation that you include in this PUT will be applied to the discussion topic.

Quizzes (Classic Quizzes)

- **Create a Quiz** (`POST /api/v1/courses/:course_id/quizzes`): Creates a new quiz in a course. **Required:** `quiz[title]` – the name of the quiz [85]. There are many optional quiz settings available: `quiz[description]` (HTML instructions), `quiz[quiz_type]` which defines the type of quiz – for a graded quiz use `"assignment"` (others include `"practice_quiz"`, `"graded_survey"`, `"survey"`) [86]. If the quiz is graded, you can assign it to an `assignment_group_id` (category) [87]. Other options include `time_limit` (in minutes) [88], `shuffle_answers` (randomize answer order) [89], and quiz scoring policies: e.g. number of attempts (`allowed_attempts`) and how to score multiple attempts (`scoring_policy`, like keep highest) [90] [91]. You can set `hide_results` to control if/when students see their quiz results (`"always"` to never show scores, etc.) [92], and related flags like `show_correct_answers` with timing for correct answer visibility [93]. Availability dates (`due_at`, `unlock_at`, `lock_at`) can be specified to integrate with the course schedule [94] [95]. If `published` is true, the quiz is available to students; if false, it's created in an unpublished state [96]. (Creating a graded quiz also creates a corresponding assignment entry automatically in Canvas.)
- **Update a Quiz** (`PUT /api/v1/courses/:course_id/quizzes/:id`): Modifies an existing quiz. Supply any of the same `quiz[...]` fields from quiz creation to change those properties [97]. For example, you can change the `title` or `description`, adjust the time limit, change the due date, or publish/unpublish the quiz by setting the `published` field. When updating, you only need to include the fields you want to change; unspecified fields will remain as they were. (Canvas notes that the update endpoint accepts the **same parameters as quiz creation** [98], so refer to the creation fields for what can be changed.) Keep in mind that some changes may be restricted if students have already taken the quiz (for instance, you cannot unpublish a quiz if there are submissions).

Note: **New Quizzes** (the Quizzes.Next feature) use a different API (`/api/quiz/v1/` paths). The above endpoints refer to classic quizzes in Canvas.

Modules

- **Create a Module** (`POST /api/v1/courses/:course_id/modules`): Creates a new module in the course content outline. **Required:** `module[name]` – the name of the module [99] [100]. Optional fields: `module[position]` to set its order (if not provided, it's appended at the end) [101], `module[unlock_at]` to schedule when the module unlocks for students (ISO datetime) [102], `module[require_sequential_progress]` (set to true to require completing items in order) [103], and `module[prerequisite_module_ids]` – an array of module IDs that must be completed before this one unlocks [104]. You can also set `module[published]` to false if you want to create

the module in an unpublished state (hidden from students) [105]. The response will include the new module's ID (needed for adding items).

- **Update a Module** (`PUT /api/v1/courses/:course_id/modules/:id`): Updates an existing module's settings. You can change the `module[name]` (title) [106], adjust `unlock_at` date, or modify the `position` in the course outline [107]. You may also toggle `require_sequential_progress` and update prerequisites by providing a new list of `prerequisite_module_ids` [104]. Additionally, setting `module[published]` to true/false will publish or unpublish the module (note: unpublishing a module will also hide its items from students) [105]. Only include the fields you want to change; others can be omitted.
- **Add a Module Item** (`POST /api/v1/courses/:course_id/modules/:module_id/items`): Adds a new item (unit of content) to a module. **Required:** `module_item[type]` – the type of item to add, and the identifier for the item. For items that point to existing course content, provide `module_item[content_id]` (e.g. the assignment ID, page ID, etc.) along with the appropriate type. For example, to add an assignment, use `type=Assignment` and `content_id=<assignment_id>` [108] [109]. If adding a wiki page, use `type=Page` and provide either `content_id` or the `page_url` of an existing page (if the page doesn't exist and you use `page_url`, a new page will be created with that URL) [109] [110]. The allowed `type` values include: `Page`, `Assignment`, `Discussion`, `Quiz`, `File`, `ExternalUrl`, `ExternalTool`, and `SubHeader` (a text heading) [111] [112]. Optional fields: `module_item[title]` (if you want a custom name different from the content's name; not required for internal items as it defaults to the content's title) [113], `module_item[position]` (position in module) [114], `module_item[indent]` (indent level for display) [115], and `module_item[completion_requirement]` to set requirements (e.g. `{"type": "must_view"}` or `must_submit`, etc., with `min_score` if applicable) [116] [117]. This will return the new Module Item object.
- **Update a Module Item** (`PUT /api/v1/courses/:course_id/modules/:module_id/items/:id`): Updates an existing module item's attributes. You can change the `module_item[title]` (for a custom label), `position` (reorder within the module), or `indent`. You may also adjust the `completion_requirement` (for instance, require a different minimum score or mark the item as optional by clearing requirements). Essentially, this uses the same fields as creation – any provided fields will overwrite the current values for that item. (For example, to mark a module item as complete for a student, a different endpoint exists; but to change the requirement type or to move the item, you'd use this PUT.) The module item ID and parent module ID are in the path.

Each endpoint above enables **programmatic creation or update** of Canvas course content and structure. Using these, a developer (for example, writing a Python script with the Canvas API) can automate course setup: creating courses, enrolling students, building modules with pages/assignments/quizzes, setting up groups, and so on – all via REST calls to Canvas.

Sources: Canvas LMS REST API Documentation [118] [4] [7] [12] [13] [19] [21] [28] [30] [119] [39] [42] [44] [45] [48] [52] [55] [56] [64] [65] [72] [74] [86] [88] [98] [99] [106] [108] [111]. (Refer to Canvas's official API docs for complete details on all parameters and data formats.)

Courses - Canvas LMS REST API Documentation
https://documentation.instructure.com/doc/api/courses.html

Assignment Groups - Canvas LMS REST API Documentation
https://www.umjicanvas.com/doc/api/assignment_groups.html

Assignments - Canvas LMS REST API Documentation
https://documentation.instructure.com/doc/api/assignments.html

Group Categories - Canvas LMS REST API Documentation
https://canvas.instructure.com/doc/api/group_categories.html

Canvas LMS REST API Documentation
http://www.humandesigncollege.org/doc/api/group_categories.html

Groups - Canvas LMS REST API Documentation
https://canvas.creatiefschrijven.be/doc/api/groups.html

Enrollments - Canvas LMS REST API Documentation
https://www.umjicanvas.com/doc/api/enrollments.html

Rubrics | Instructure Developer Documentation Portal
https://developerdocs.instructure.com/services/canvas/file.all_resources/rubrics

Pages - Canvas LMS REST API Documentation
https://canvas.creatiefschrijven.be/doc/api/pages.html

Course — canvasapi 3.2.0 documentation
https://canvasapi.readthedocs.io/en/stable/course-ref.html

Discussion Topics - Canvas LMS REST API Documentation
https://canvas.colorado.edu/doc/api/discussion_topics.html

Canvas LMS REST API Documentation
https://mitt.uib.no/doc/api/all_resources.html

Modules | Instructure Developer Documentation Portal

https://developerdocs.instructure.com/services/canvas/file.all_resources/modules