Comprehensive Redmine REST API Documentation

Stable APIs for Building Full-Fledged Applications

This documentation covers all **STABLE** APIs available in Redmine for building robust applications using AI tools or traditional development.

Table of Contents

1. Authentication
2. General Guidelines
3. Issues API
4. Projects API
5. Users API
6. Time Entries API
7. Project Memberships API
8. News API
9. Issue Relations API
10. Versions API
11. Wiki Pages API
12. Queries API
13. Attachments API
14. Issue Statuses API
15. Trackers API
16. Enumerations API
17. Issue Categories API
18. Roles API
19. Groups API
20. Custom Fields API
21. Search API
22. Files API

23. My Account API
24. Journals API

Authentication

Enabling the API

In Redmine Administration → Settings → API, check "**Enable REST API**".

Authentication Methods

1. API Key (Recommended)

- Find your API key at `/my/account` (right pane when logged in)
- Three ways to pass the API key:

```bash
# As query parameter
GET /issues.json?key=YOUR_API_KEY

# As HTTP header (recommended)
GET /issues.json
X-Redmine-API-Key: YOUR_API_KEY

# As username in HTTP Basic Auth
GET /issues.json
Authorization: Basic BASE64(YOUR_API_KEY:random_password)
```

2. HTTP Basic Authentication

```bash
Authorization: Basic BASE64(username:password)
```

3. User Impersonation (Admin only)

```bash
X-Redmine-Switch-User: username
```

Returns 412 error if user doesn't exist or is inactive.

General Guidelines

Content Types

Always specify `Content-Type` header for POST/PUT requests:

- **JSON**: `Content-Type: application/json`
- **XML**: `Content-Type: application/xml`

Response Formats

Append `.json` or `.xml` to URLs:

```bash
GET /issues.json
GET /issues.xml
```

Pagination

Default: 25 items, Maximum: 100 items per page

```bash
GET /issues.json?offset=0&limit=100
GET /issues.json?offset=100&limit=100
```

Response includes:

```json
{
  "issues": [...],
  "total_count": 2595,
  "limit": 25,
  "offset": 0
}
```

To disable metadata:

```bash
GET /issues.json?nometa=1
```

Including Associated Data

Use the `include` parameter:

bash

```
# Single association
GET /issues/296.json?include=journals

# Multiple associations
GET /issues/296.json?include=journals,attachments,relations
```

Custom Fields

Custom fields appear in responses:

json

```
{
  "custom_fields": [
    {"id": 1, "name": "Version", "value": "1.0.1"},
    {"id": 2, "name": "Resolution", "value": "Fixed"}
  ]
}
```

To set/update custom fields:

json

```
{
  "issue": {
    "custom_fields": [
      {"id": 1, "value": "1.0.2"},
      {"id": 2, "value": "Invalid"}
    ]
  }
}
```

Error Handling

422 Unprocessable Entity - Validation errors:

json

```
{
  "errors": [
    "First name can't be blank",
    "Email is invalid"
  ]
}
```

Issues API

List Issues

```bash
GET /issues.json
```

Parameters:

- `offset`, `limit` - Pagination
- `sort` - Column to sort (append `:desc` for descending)
- `issue_id` - Get specific issues: `?issue_id=1,2,3`
- `project_id` - Filter by project ID or identifier
- `subproject_id` - Include subprojects
- `tracker_id` - Filter by tracker ID
- `status_id` - Filter by status (`open`, `closed`, `*`, or status ID)
- `assigned_to_id` - Filter by assigned user (use `me` for current user)
- `parent_id` - Filter by parent issue
- `cf_x` - Filter by custom field (x = custom field ID)
- `created_on`, `updated_on` - Date filters (operators: `>=`, `<=`, `><`)

Date Filter Examples:

```bash
# Date range
GET /issues.json?created_on=><2012-03-01|2012-03-07

# After date
GET /issues.json?created_on=>=2012-03-01

# Before date
GET /issues.json?created_on=<=2012-03-07

# Timestamp
GET /issues.json?updated_on=>=2014-01-02T08:12:32Z
```

Include Options:

- `attachments`, `relations`, `journals`, `changesets`, `watchers`, `children`, `allowed_statuses`

Get Single Issue

```bash
GET /issues/123.json
GET /issues/123.json?include=attachments,journals
```

Create Issue

```bash
POST /issues.json
Content-Type: application/json

{
  "issue": {
    "project_id": 1,
    "tracker_id": 1,
    "status_id": 1,
    "priority_id": 4,
    "subject": "Issue subject",
    "description": "Issue description",
    "category_id": 5,
    "fixed_version_id": 3,
    "assigned_to_id": 2,
    "parent_issue_id": 10,
    "custom_fields": [
      {"id": 1, "value": "1.0.2"}
    ],
    "watcher_user_ids": [3, 5],
    "uploads": [
      {
        "token": "7167.ed1ccdb093229ca1bd0b043618d88743",
        "filename": "file.pdf",
        "content_type": "application/pdf"
      }
    ]
  }
}
```

Response: 201 Created with issue details

Update Issue

```bash
```

```
PUT /issues/123.json
Content-Type: application/json

{
  "issue": {
    "subject": "Updated subject",
    "notes": "Update comment",
    "status_id": 5,
    "priority_id": 6
  }
}
```

Response: 204 No Content (success) or 422 Unprocessable Entity (validation error)

Delete Issue

```
bash

DELETE /issues/123.json
```

Response: 204 No Content

Add Watcher

```
bash

POST /issues/123/watchers.json
Content-Type: application/json

{
  "user_id": 3
}
```

Remove Watcher

```
bash

DELETE /issues/123/watchers/3.json
```

Projects API

List Projects

```
bash

GET /projects.json
```

GET /projects.json

Parameters:

- include - Options: trackers, issue_categories, enabled_modules, time_entry_activities, issue_custom_fields

Get Project

```bash
GET /projects/123.json
GET /projects/project-identifier.json
GET /projects/123.json?include=trackers,issue_categories
```

Include Options:

- trackers, issue_categories, enabled_modules, time_entry_activities, issue_custom_fields

Response:

```json
{
  "project": {
    "id": 1,
    "name": "Project Name",
    "identifier": "project-identifier",
    "description": "Description",
    "status": 1,
    "is_public": true,
    "created_on": "2007-09-29T12:03:04Z",
    "updated_on": "2009-03-15T12:35:11Z",
    "custom_fields": [...]
  }
}
```

Create Project

```bash
```

```
POST /projects.json
Content-Type: application/json

{
  "project": {
    "name": "New Project",
    "identifier": "new-project",
    "description": "Project description",
    "is_public": true,
    "parent_id": 1,
    "inherit_members": true,
    "tracker_ids": [1, 2],
    "enabled_module_names": ["issue_tracking", "time_tracking", "wiki"]
  }
}
```

Update Project

```bash
PUT /projects/123.json
Content-Type: application/json

{
  "project": {
    "name": "Updated Name",
    "description": "Updated description"
  }
}
```

Delete Project

```bash
DELETE /projects/123.json
```

Users API

Note: Most operations require admin privileges

List Users

```bash
```

```
GET /users.json
```

Parameters:

- `status` - Filter by status (1=Active, 2=Registered, 3=Locked)
- `name` - Filter by login, firstname, lastname, or email
- `group_id` - Filter by group membership

Get User

```bash
GET /users/123.json
GET /users/123.json?include=groups,memberships
```

Include Options:

- `groups`, `memberships`

Create User (Admin only)

```bash
POST /users.json
Content-Type: application/json

{
  "user": {
    "login": "jsmith",
    "password": "secret",
    "firstname": "John",
    "lastname": "Smith",
    "mail": "jsmith@example.com",
    "auth_source_id": 1,
    "mail_notification": "all",
    "must_change_passwd": false,
    "generate_password": false
  }
}
```

Update User (Admin only)

```bash
PUT /users/123.json
Content-Type: application/json
```

```
{
  "user": {
    "firstname": "Jane",
    "mail": "jane@example.com"
  }
}
```

Delete User (Admin only)

```
bash

DELETE /users/123.json
```

Time Entries API

List Time Entries

```
bash

GET /time_entries.json
```

Parameters:

- `user_id` - Filter by user ID
- `project_id` - Filter by project ID or identifier
- `spent_on` - Filter by date (supports operators: `><`, `>=`, `<=`)
- `from`, `to` - Date range filter

Examples:

```
bash

GET /time_entries.json?user_id=5
GET /time_entries.json?project_id=project-identifier
GET /time_entries.json?spent_on=><2013-05-01|2013-05-31
```

Get Time Entry

```
bash

GET /time_entries/123.json
```

Create Time Entry

Create Time Entry

```bash
POST /time_entries.json
Content-Type: application/json

{
  "time_entry": {
    "issue_id": 123,
    "spent_on": "2020-12-24",
    "hours": 3.5,
    "activity_id": 8,
    "comments": "Time entry comment",
    "user_id": 2
  }
}
```

Note: `project_id` OR `issue_id` is required. If `activity_id` is not provided, default activity is used.

Response: 201 Created

Update Time Entry

```bash
PUT /time_entries/123.json
Content-Type: application/json

{
  "time_entry": {
    "hours": 4.0,
    "comments": "Updated comment"
  }
}
```

Delete Time Entry

```bash
```

```
DELETE /time_entries/123.json
```

Project Memberships API

List Memberships

bash

```
GET /projects/123/memberships.json
GET /projects/project-identifier/memberships.json
```

Response:

json


```
{
  "memberships": [
    {
      "id": 1,
      "project": {"id": 1, "name": "Project"},
      "user": {"id": 17, "name": "John Smith"},
      "roles": [
        {"id": 1, "name": "Manager"}
      ]
    },
    {
      "id": 3,
      "project": {"id": 1, "name": "Project"},
      "group": {"id": 24, "name": "Contributors"},
      "roles": [
        {"id": 3, "name": "Contributor"}
      ]
    }
  ],
  "total_count": 2,
  "offset": 0,
  "limit": 25
}
```


Get Membership

```bash
GET /memberships/123.json
```

Add Member

```bash
POST /projects/123/memberships.json
Content-Type: application/json

{
  "membership": {
    "user_id": 27,
    "role_ids": [2, 3]
  }
}
```

Response: 201 Created

Update Membership

```bash
PUT /memberships/123.json
Content-Type: application/json

{
  "membership": {
    "role_ids": [3, 4]
  }
}
```

Delete Membership

```bash
DELETE /memberships/123.json
```

Note: Inherited memberships (from groups) cannot be deleted directly.

News API

Note: Currently only GET (index) is stable

List News

```bash
GET /news.json
GET /projects/123/news.json
```

Issue Relations API

List Relations for Issue

```bash
GET /issues/123/relations.json
```

Response:

```json
{
  "relations": [
    {
      "id": 1819,
      "issue_id": 8470,
      "issue_to_id": 8469,
      "relation_type": "relates",
      "delay": null
    }
  ]
}
```

Relation Types:

- `relates`, `duplicates`, `duplicated`, `blocks`, `blocked`, `precedes`, `follows`, `copied_to`, `copied_from`

Get Relation

```bash
```

```
GET /relations/1819.json
```

Create Relation

```bash
POST /issues/123/relations.json
Content-Type: application/json

{
  "relation": {
    "issue_to_id": 456,
    "relation_type": "relates",
    "delay": 0
  }
}
```

Response: 201 Created

Delete Relation

```bash
DELETE /relations/1819.json
```

Versions API

List Versions

```bash
GET /projects/123/versions.json
GET /projects/project-identifier/versions.json
```

Response:

```json
{
  "versions": [
    {
      "id": 1,
      "project": {"id": 1, "name": "Project"},
      "name": "1.0",
```

```
    "description": "Version description",
    "status": "open",
    "due_date": "2024-12-31",
    "sharing": "none",
    "wiki_page_title": "Version_1_0",
    "created_on": "2023-01-01T00:00:00Z",
    "updated_on": "2023-06-01T00:00:00Z"
   }
  ],
  "total_count": 1
}
```


Status Values: open, locked, closed

Sharing Values: none, descendants, hierarchy, tree, system

Get Version

bash

```
GET /versions/123.json
```

Create Version

bash

```
POST /projects/123/versions.json
Content-Type: application/json

{
  "version": {
    "name": "2.0",
    "description": "Next major release",
    "status": "open",
    "due_date": "2025-06-30",
    "sharing": "none",
    "wiki_page_title": "Version_2_0"
```

```
  }
}
```

Response: 201 Created

Update Version

```bash
PUT /versions/123.json
Content-Type: application/json

{
  "version": {
    "name": "2.0.1",
    "status": "closed"
  }
}
```

Delete Version

```bash
DELETE /versions/123.json
```

Wiki Pages API

List Wiki Pages

```bash
GET /projects/project-identifier/wiki/index.json
```

Response:

```json
```

```
{
  "wiki_pages": [
    {
      "title": "UsersGuide",
      "version": 2,
      "created_on": "2008-03-09T12:07:08Z",
      "updated_on": "2008-03-09T23:41:33Z"
    }
  ]
}
```


Get Wiki Page

```
bash

GET /projects/project-identifier/wiki/PageName.json
GET /projects/project-identifier/wiki/PageName.json?include=attachments
```

Get Specific Version:

```
bash

GET /projects/project-identifier/wiki/PageName/2.json
```

Response:


```
json

{
  "wiki_page": {
    "title": "PageName",
    "text": "h1. Page Content...",
    "version": 22,
    "author": {"id": 11, "name": "John Smith"},
    "comments": "Updated",
    "created_on": "2009-05-18T20:11:52Z",
    "updated_on": "2012-10-02T11:38:18Z"
  }
}
```


Create/Update Wiki Page

```
bash

PUT /projects/project-identifier/wiki/PageName.json
Content-Type: application/json
```

```
{
  "wiki_page": {
    "text": "h1. Page Title\n\nPage content here",
    "comments": "Update comment",
    "version": 18
  }
}
```

With Attachments:

```bash
# First upload files
POST /uploads.json?filename=image.png
Content-Type: application/octet-stream
[binary file content]

# Response: {"upload":{"token":"7167.ed1..."}}

# Then create/update wiki page
PUT /projects/project-identifier/wiki/PageName.json
Content-Type: application/json

{
  "wiki_page": {
    "text": "Content with !image.png!",
    "uploads": [
      {
        "token": "7167.ed1...",
        "filename": "image.png",
        "content_type": "image/png"
      }
    ]
  }
}
```

Note: The `text` field is required. Include `version` for optimistic locking (returns 409 Conflict if version mismatch).

Delete Wiki Page

```bash
DELETE /projects/project-identifier/wiki/PageName.json
```

Queries API

List Queries

```bash
GET /queries.json
```

Response:

```json
{
  "queries": [
    {
      "id": 84,
      "name": "Documentation issues",
      "is_public": true,
      "project_id": 1
    }
  ],
  "total_count": 5,
  "offset": 0,
  "limit": 25
}
```

Use Query to Get Issues

```bash
GET /issues.json?query_id=84
GET /issues.json?query_id=84&project_id=foo
```

Attachments API

Get Attachment

```bash
GET /attachments/123.json
```

Response:

```json
```

```
{
  "attachment": {
    "id": 6243,
    "filename": "document.pdf",
    "filesize": 124567,
    "content_type": "application/pdf",
    "description": "Important document",
    "content_url": "http://redmine/attachments/download/6243/document.pdf",
    "author": {"id": 1, "name": "John Smith"},
    "created_on": "2011-07-18T22:58:40Z"
  }
}
```


Update Attachment

bash

```
PATCH /attachments/123.json
Content-Type: application/json

{
  "attachment": {
    "description": "Updated description"
  }
}
```

Download Attachment

Use the `content_url` from the attachment response to download the file.

Upload and Attach Files

Step 1: Upload File

bash

```
POST /uploads.json?filename=document.pdf
Content-Type: application/octet-stream
[binary file content]
```

Response:

```
json

{
  "upload": {
    "token": "7167.ed1ccdb093229ca1bd0b043618d88743"
  }
}
```

Step 2: Attach to Resource (Issue, Wiki, etc.)

```
bash

# When creating/updating an issue
POST /issues.json
{
  "issue": {
    "project_id": 1,
    "subject": "Issue with attachment",
    "uploads": [
      {
        "token": "7167.ed1ccdb093229ca1bd0b043618d88743",
        "filename": "document.pdf",
        "content_type": "application/pdf",
        "description": "Optional description"
      }
    ]
  }
}
```

Note: Include attachments with issues using: `GET /issues/123.json?include=attachments`

Issue Statuses API

List Issue Statuses

```
bash

GET /issue_statuses.json
```

Response:

```json
{
  "issue_statuses": [
    {
      "id": 1,
      "name": "New",
      "is_closed": false
    },
    {
      "id": 5,
      "name": "Closed",
      "is_closed": true
    }
  ]
}
```

Trackers API

List Trackers

```bash
GET /trackers.json
```

Response:

```json
{
  "trackers": [
    {
      "id": 1,
      "name": "Bug",
      "default_status": {"id": 1, "name": "New"},
      "description": "Bug tracker description",
      "enabled_standard_fields": [
        "assigned_to_id",
        "category_id",
        "fixed_version_id",
        "parent_issue_id",
        "start_date",
        "due_date",
        "estimated_hours",
```

```
      "done_ratio",
      "description"
    ]
  }
 ]
}
```

Enumerations API

Issue Priorities

bash

```
GET /enumerations/issue_priorities.json
```

Response:

json

```
{
 "issue_priorities": [
  {
   "id": 3,
   "name": "Low",
   "is_default": false
  },
  {
   "id": 4,
   "name": "Normal",
   "is_default": true
  },
  {
   "id": 5,
   "name": "High",
   "is_default": false
  }
 ]
}
```

Time Entry Activities

bash

```
GET /enumerations/time_entry_activities.json
```

Response:

```
json

{
  "time_entry_activities": [
    {
      "id": 8,
      "name": "Design",
      "is_default": false
    },
    {
      "id": 9,
      "name": "Development",
      "is_default": true
    }
  ]
}
```

Document Categories

```
bash


GET /enumerations/document_categories.json
```

Issue Categories API

List Categories

```
bash

GET /projects/123/issue_categories.json
```

Response:

```
json

{
  "issue_categories": [
    {
      "id": 57,
      "project": {"id": 17, "name": "Project"},
```

```
    "name": "UI",
    "assigned_to": {"id": 22, "name": "John Smith"}
   }
  ],
  "total_count": 2
}
```


Get Category

bash

```
GET /issue_categories/57.json
```

Create Category

bash

```
POST /projects/123/issue_categories.json
Content-Type: application/json

{
  "issue_category": {
    "name": "Backend",
    "assigned_to_id": 5
  }
}
```

Response: 201 Created

Update Category

bash

```
PUT /issue_categories/57.json
Content-Type: application/json

{
  "issue_category": {
    "name": "Frontend",
    "assigned_to_id": 10
  }
}
```

Delete Category

bash

```
DELETE /issue_categories/57.json
```

DELETE /issue_categories/37.json

Roles API

List Roles

```
bash

GET /roles.json
```

Response:

```
json
```


```
{
  "roles": [
    {
      "id": 1,
      "name": "Manager"
    },
    {
      "id": 2,
      "name": "Developer"
    }
  ]
}
```


Get Role (with permissions)

```
bash

GET /roles/5.json
```

Response:

```json
{
  "role": {
    "id": 5,
    "name": "Reporter",
    "assignable": true,
    "issues_visibility": "default",
    "time_entries_visibility": "all",
    "users_visibility": "all",
    "permissions": [
      "view_issues",
      "add_issues",
      "edit_issues",
      "add_issue_notes"
    ]
  }
}
```

Groups API

Note: Requires admin privileges

List Groups

```bash
GET /groups.json
```

Response:

```json
{
  "groups": [
    {
      "id": 53,
      "name": "Managers"
    },
```

```
    {
      "id": 55,
      "name": "Developers"
    }
  ]
}
```

Get Group

```
bash

GET /groups/53.json
```

Response:

```
json

{
  "group": {
    "id": 20,
    "name": "Developers",
    "users": [
      {"id": 5, "name": "John Smith"},
      {"id": 8, "name": "Dave Loper"}
    ]
  }
}
```

Create Group

```
bash

POST /groups.json
Content-Type: application/json


{
  "group": {
    "name": "QA Team",
    "user_ids": [3, 5, 7]
  }
}
```

Response: 201 Created

Update Group

```
bash
```

```
PUT /groups/53.json
Content-Type: application/json

{
  "group": {
    "name": "Senior Developers",
    "user_ids": [5, 8, 12]
  }
}
```

Delete Group

```bash
DELETE /groups/53.json
```

Add User to Group

```bash
POST /groups/53/users.json
Content-Type: application/json

{
  "user_id": 15
}
```

Remove User from Group

```bash
DELETE /groups/53/users/15.json
```

Custom Fields API

Note: Requires admin privileges

List Custom Fields

```bash
GET /custom_fields.json
```

Response:

```json
{
  "custom_fields": [
    {
      "id": 1,
      "name": "Affected Version",
      "customized_type": "issue",
      "field_format": "list",
      "regexp": "",
      "min_length": null,
      "max_length": null,
      "is_required": true,
      "is_filter": true,
      "searchable": true,
      "multiple": true,
      "default_value": "",
      "visible": false,
      "possible_values": [
        {"value": "0.5.x"},
        {"value": "0.6.x"},
        {"value": "1.0.x"}
      ]
    }
  ]
}
```

Customized Types:

- `issue`, `time_entry`, `project`, `version`, `user`, `group`, `issue_priority`, `document_category`

Field Formats:

- `string`, `text`, `int`, `float`, `list`, `date`, `bool`, `user`, `version`, `link`, `attachment`

Search API

Search

```bash
GET /search.json?q=search_terms
```

Parameters:

- `q` - Search query (space-separated terms)

- `offset`, `limit` - Pagination
- `scope` - Search scope: `all`, `my_projects`, `subprojects`
- `all_words` - Match all words (boolean)
- `titles_only` - Search only titles (boolean)
- Filters: `issues`, `news`, `documents`, `changesets`, `wiki_pages`, `messages`, `projects`

Examples:

```bash
# Search all
GET /search.json?q=database+optimization

# Search issues and wikis only
GET /search.json?q=bug&issues=1&wiki_pages=1

# With pagination
GET /search.json?q=query&offset=0&limit=100
```

Response:

```json
{
```

```
  "results": [
    {
      "id": 5,
      "title": "Wiki: Configuration_Guide",
      "type": "wiki-page",
      "url": "http://redmine/projects/foo/wiki/Configuration_Guide",
      "description": "h1. Configuration..."
    },
    {
      "id": 123,
      "title": "#123: Bug in login",
      "type": "issue",
      "url": "http://redmine/issues/123",
      "description": "Description..."
    }
  ],
  "total_count": 45,
  "offset": 0,
  "limit": 25
}
```

Files API

List Files

```
bash

GET /projects/123/files.json
GET /projects/project-identifier/files.json
```

Response:


```
json

{
  "files": [
    {
      "id": 12,
      "filename": "app-1.0-setup.exe",
      "filesize": 74753799,
      "content_type": "application/octet-stream",
      "description": "Application installer",
      "content_url": "http://redmine/attachments/download/12/app-1.0-setup.exe",
      "author": {"id": 1, "name": "Admin"},
      "created_on": "2017-01-04T09:12:32Z",
      "version": {"id": 2, "name": "1.0"},
```

```
    "digest": "1276481102f218c981e0324180bafd9f",
    "downloads": 142
   }
 ]
}
```


Add File

```bash
# Step 1: Upload file
POST /uploads.json?filename=release-2.0.zip
Content-Type: application/octet-stream
[binary content]

# Response: {"upload": {"token": "21.01a1d7..."}}

# Step 2: Create file entry
POST /projects/123/files.json
Content-Type: application/json

{
  "file": {
    "token": "21.01a1d7b1c2ffcbbc9ecf14debeec27d8",
    "version_id": 2,
    "filename": "release-2.0.zip",
    "description": "Release 2.0 package"
  }
}
```

My Account API

Get My Account

```bash
```

```
GET /my/account.json
```

Response:

```
json

{
  "user": {
    "id": 3,
    "login": "jsmith",
    "admin": false,
    "firstname": "John",
    "lastname": "Smith",
    "mail": "jsmith@example.com",
    "created_on": "2006-07-19T17:33:19Z",
    "last_login_on": "2020-06-14T13:03:34Z",
    "api_key": "c308a59c9dea95920b13522fb3e0fb7fae4f292d",
    "custom_fields": [
      {
        "id": 4,
        "name": "Phone",
        "value": "+1234567890"
      }
    ]
  }
}
```

Update My Account

```
bash

PUT /my/account.json
Content-Type: application/json


{
  "user": {
    "firstname": "Jane",
    "mail": "jane@example.com",
    "custom_fields": [
      {"id": 4, "value": "+9876543210"}
    ]
  }
}
```

Journals API

Journals API

Journals track issue history (comments and changes). They're accessed via the Issues API.

Get Issue with Journals

```bash
GET /issues/123.json?include=journals
```

Response:

```json
{
  "issue": {
    "id": 123,
    "subject": "Issue subject",
    "journals": [
      {
        "id": 1,
        "user": {"id": 1, "name": "John Smith"},
        "notes": "Fixed in Revision 128",
        "created_on": "2007-01-01T05:21:00Z",
        "private_notes": false,
        "details": []
      },
      {
        "id": 2,
        "user": {"id": 5, "name": "Jane Doe"},
```

```
      "notes": "",
      "created_on": "2009-08-13T11:33:17Z",
      "details": [
       {
         "property": "attr",
         "name": "status_id",
         "old_value": "1",
         "new_value": "5"
       }
      ]
    }
  ]
 }
}
```


Add Note to Issue

```bash
PUT /issues/123.json
Content-Type: application/json

{
  "issue": {
    "notes": "This is a comment on the issue"
  }
}
```

Note: You can include `private_notes: true` to make the note private (visible only to users with proper permissions).

Advanced Usage Patterns

Filtering with Custom Fields

```bash
# Filter issues by custom field (cf_1 = custom field ID 1)
GET /issues.json?cf_1=value

# Multiple custom field filters
GET /issues.json?cf_1=value1&cf_2=value2

# Custom field with multiple values (use pipe)
GET /issues.json?cf_1=value1|value2
```

Complex Issue Queries

```bash
# Get open issues assigned to me with high priority
GET /issues.json?assigned_to_id=me&status_id=open&priority_id=5

# Get issues in specific project with tracker and custom field
GET /issues.json?project_id=project-id&tracker_id=1&cf_1=value

# Get issues updated in last 7 days
GET /issues.json?updated_on=>=2024-01-01T00:00:00Z

# Combine filters with includes
GET /issues.json?project_id=1&status_id=open&include=attachments,watchers
```

Batch Operations

```bash
# Get multiple specific issues
GET /issues.json?issue_id=1,2,3,4,5&limit=100

# Process in batches
GET /issues.json?offset=0&limit=100
GET /issues.json?offset=100&limit=100
GET /issues.json?offset=200&limit=100
```

Working with Attachments

```bash
```

```
# 1. Upload multiple files
POST /uploads.json?filename=file1.pdf
POST /uploads.json?filename=file2.png

# 2. Create issue with multiple attachments
POST /issues.json
{
  "issue": {
    "project_id": 1,
    "subject": "Issue with documents",
    "uploads": [
      {"token": "token1", "filename": "file1.pdf", "content_type": "application/pdf"},
      {"token": "token2", "filename": "file2.png", "content_type": "image/png"}
    ]
  }
}

# 3. Get issue with attachments
GET /issues/123.json?include=attachments
```

Building Complete Applications

User Story: Project Management Dashboard

1. **Get all active projects**

```
bash

GET /projects.json?include=trackers,issue_categories
```

2. **For each project, get open issues**

```
bash

GET /issues.json?project_id={project_id}&status_id=open&include=attachments,watchers
```

3. **Get team members**

```
bash

GET /projects/{project_id}/memberships.json
```

4. **Get time entries for reporting**

bash

```
GET /time_entries.json?project_id={project_id}&spent_on=><2024-01-01|2024-01-31
```

5. **Get project versions/milestones**

bash

```
GET /projects/{project_id}/versions.json
```

User Story: Issue Tracking Integration

1. **Authenticate user**

bash

```
GET /my/account.json
```

2. **List available trackers and statuses**

bash

```
GET /trackers.json
GET /issue_statuses.json
GET /enumerations/issue_priorities.json
```

3. **Get custom fields for issue creation**

bash

```
GET /custom_fields.json
```

4. **Create issue with all metadata**

bash

```
POST /issues.json
{
  "issue": {
    "project_id": 1,
    "tracker_id": 1,
    "status_id": 1,
    "priority_id": 4,
    "subject": "New bug",
    "description": "Description",
    "custom_fields": [...]
  }
}
```

5. **Monitor issue changes**

```
bash

GET /issues/{id}.json?include=journals
```

Error Codes Reference

Code	Meaning	Common Causes
200	OK	Successful GET request
201	Created	Successful POST (creation)
204	No Content	Successful PUT/DELETE
401	Unauthorized	Missing or invalid API key
403	Forbidden	Insufficient permissions
404	Not Found	Resource doesn't exist
406	Not Acceptable	Missing Content-Type header
409	Conflict	Version conflict (wiki pages)
412	Precondition Failed	Invalid user in X-Redmine-Switch-User
422	Unprocessable Entity	Validation errors
500	Internal Server Error	Server-side error

Best Practices

1. Authentication

- Use API key in header (`X-Redmine-API-Key`) rather than URL parameter
- Generate separate API keys for different applications
- Never commit API keys to version control

2. Performance

- Always use pagination for large datasets
- Use `include` parameter to reduce number of requests
- Cache enumerations (statuses, trackers, priorities) as they rarely change
- Implement rate limiting in your application

3. Error Handling