Redmine API

Redmine exposes some of its data through a REST API. This API provides access and basic CRUD operations (create, update, delete) for the resources described below. The API supports both XML and JSON formats.

API Description

Resource	Status	Notes	Availability
Issues	Stable		1.0
Projects	Stable		1.0
Project Memberships	Alpha		1.4
Users	Stable		1.1
Time Entries	Stable		1.1
News	Prototype	Prototype implementation for `index` only	1.1
Issue Relations	Alpha		1.3
Versions	Alpha		1.3
Wiki Pages	Alpha		2.2
Queries	Alpha		1.3
Attachments	Beta	Adding attachments via the API added in 1.4	1.3
Issue Statuses	Alpha	Provides the list of all statuses	1.3
Trackers	Alpha	Provides the list of all trackers	1.3
Enumerations	Alpha	Provides the list of issue priorities and time tracking activities	2.2
Issue Categories	Alpha		1.3
Roles	Alpha		1.4
Groups	Alpha		2.1
Custom Fields	Alpha		2.4

Search	Alpha		3.3
Files	Alpha		3.4

Status legend:

- Stable - feature complete, no major changes planned
- Beta - usable for integrations with some bugs or missing minor functionality
- Alpha - major functionality in place, needs feedback from API users and integrators
- Prototype - very rough implementation, possible major breaking changes mid-version. **Not recommended for integration**
- Planned - planned in a future version, depending on developer availability

You can review the list of all the API changes for each version.

General topics

Specify `Content-Type` on `POST`/`PUT` requests

When creating or updating a remote element, the `Content-Type` of the request **MUST** be specified even if the remote URL is suffixed accordingly (e.g. `POST ../issues.json`):

- for JSON content, it must be set to `Content-Type: application/json`.
- for XML content, to `Content-Type: application/xml`.

Authentication

Most of the time, the API requires authentication. To enable the API-style authentication, you have to check **Enable REST API** in Administration -> Settings -> API. Then, authentication can be done in 2 different ways:

- using your regular login/password via HTTP Basic authentication.
- using your API key which is a handy way to avoid putting a password in a script. The API key may be attached to each request in one of the following way:
  - passed in as a "key" parameter
  - passed in as a username with a random password via HTTP Basic authentication
  - passed in as a "X-Redmine-API-Key" HTTP header (added in Redmine 1.1.0)

You can find your API key on your account page (/my/account) when logged in, on the right-hand pane of the default layout.

User Impersonation

As of Redmine 2.2.0, you can impersonate user through the REST API by setting the `X-Redmine-Switch-User` header of your API request. It must be set to a user login (eg. `X-Redmine-Switch-User: jsmith`). This only works when using the API with an administrator account, this header will be ignored when using the API with a regular user account.

If the login specified with the `X-Redmine-Switch-User` header does not exist or is not active, you will receive a 412 error response.

Collection resources and pagination

The response to a GET request on a collection resources (eg. `/issues.xml`, `/users.xml`) generally won't return all the objects available in your database. Redmine 1.1.0 introduces a common way to query such resources using the following parameters:

- `offset`: the offset of the first object to retrieve
- `limit`: the number of items to be present in the response (default is 25, maximum is 100)

Examples:

```
GET /issues.xml

=> returns the 25 first issues



GET /issues.xml?limit=100

=> returns the 100 first issues



GET /issues.xml?offset=30&limit=10

=> returns 10 issues from the 30th
```

Responses to GET requests on collection resources provide information about the total object count available in Redmine and the offset/limit used for the response. Examples:

```
GET /issues.xml



<issues type="array" total_count="2595" limit="25" offset="0">

  ...

</issues>

GET /issues.json



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

Note: if you're using a REST client that does not support such top level attributes (total_count, limit, offset), you can set the `nometa` parameter or `X-Redmine-Nometa` HTTP header to 1 to get responses without them. Example:

```
GET /issues.xml?nometa=1



<issues type="array">

  ...

</issues>
```

Fetching associated data

Since of 1.1.0, you have to explicitly specify the associations you want to be included in the query result by appending the `include` parameter to the query url :

Example:

To retrieve issue journals with its description:

```
GET /issues/296.xml?include=journals



<issue>

  <id>296</id>

  ...

  <journals type="array">

  ...

  </journals>

</issue>
```

You can also load multiple associations using a comma separated list of items.

Example:

```
GET /issues/296.xml?include=journals,changesets



<issue>

  <id>296</id>

  ...

  <journals type="array">

  ...

  </journals>

  <changesets type="array">

  ...

  </changesets>

</issue>
```

Working with custom fields

Most of the Redmine objects support custom fields. Their values can be found in the `custom_fields` attributes.

XML Example:

```
GET /issues/296.xml      # an issue with 2 custom fields


<issue>

  <id>296</id>

  ...

  <custom_fields type="array">

    <custom_field name="Affected version" id="1">

      <value>1.0.1</value>

    </custom_field>

    <custom_field name="Resolution" id="2">

      <value>Fixed</value>

    </custom_field>

  </custom_fields>

</issue>
```

JSON Example:

```
GET /issues/296.json      # an issue with 2 custom fields


{"issue":

  {

    "id":8471,

    ...

    "custom_fields":

      [

        {"value":"1.0.1","name":"Affected version","id":1},

        {"value":"Fixed","name":"Resolution","id":2}

      ]

  }

}
```

You can also set/change the values of the custom fields when creating/updating an object using the same syntax (except that the custom field name is not required).

XML Example:

```
PUT /issues/296.xml


<issue>

  <subject>Updating custom fields of an issue</subject>

  ...

  <custom_fields type="array">

    <custom_field id="1">

      <value>1.0.2</value>

    </custom_field>

    <custom_field id="2">

      <value>Invalid</value>

    </custom_field>

  </custom_fields>

</issue>
```

Note: the `type="array"` attribute on `custom_fields` XML tag is strictly required.

JSON Example:

```
PUT /issues/296.json


{"issue":

  {

    "subject":"Updating custom fields of an issue",

    ...

    "custom_fields":

      [

        {"value":"1.0.2","id":1},

        {"value":"Invalid","id":2}

      ]
  }

}
```

Attaching files

Support for adding attachments through the REST API is added in Redmine 1.4.0.

First, you need to upload each file with a POST request to `/uploads.xml` (or `/uploads.json`). The request body should be the content of the file you want to attach and the `Content-Type` header must be set to `application/octet-stream` (otherwise you'll get a `406 Not Acceptable` response). If the upload succeeds, you get a 201 response that contains a token for your uploaded file.

Then you can use this token to attach your uploaded file to a new or an existing issue.

XML Example

First, upload your file:

```
POST /uploads.xml

Content-Type: application/octet-stream

...

(request body is the file content)



# 201 response

<upload>

  <token>7167.ed1ccdb093229ca1bd0b043618d88743</token>

</upload>
```

Then create the issue using the upload token:

```
POST /issues.xml

<issue>

  <project_id>1</project_id>

  <subject>Creating an issue with a uploaded file</subject>

  <uploads type="array">

    <upload>

      <token>7167.ed1ccdb093229ca1bd0b043618d88743</token>

      <filename>image.png</filename>

      <description>An optional description here</description>
      <content_type>image/png</content_type>

    </upload>

  </uploads>

</issue>
```

If you try to upload a file that exceeds the maximum size allowed, you get a 422 response:

```
POST /uploads.xml

Content-Type: application/octet-stream

...

(request body larger than the maximum size allowed)



# 422 response

<errors>

  <error>This file cannot be uploaded because it exceeds the maximum allowed file size (1024000)</error>

</errors>
```

JSON Example

First, upload your file:

```
POST /uploads.json

Content-Type: application/octet-stream

...

(request body is the file content)



# 201 response

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

Then create the issue using the upload token:

```
POST /issues.json

{

  "issue": {

    "project_id": "1",
    "subject": "Creating an issue with a uploaded file",

    "uploads": [

      {"token": "7167.ed1ccdb093229ca1bd0b043618d88743", "filename": "image.png",
"content_type": "image/png"}

    ]

  }

}
```

You can also upload multiple files (by doing multiple POST requests to `/uploads.json`), then create an issue with multiple attachments:

```
POST /issues.json

{

  "issue": {

    "project_id": "1",

    "subject": "Creating an issue with a uploaded file",

    "uploads": [

      {"token": "7167.ed1ccdb093229ca1bd0b043618d88743", "filename":
"image1.png", "content_type": "image/png"},

      {"token": "7168.d595398bbb104ed3bba0eed666785cc6", "filename":
"image2.png", "content_type": "image/png"}

    ]

  }

}
```

Validation errors

When trying to create or update an object with invalid or missing attribute parameters, you will get a `422 Unprocessable Entity` response. That means that the object could not be created or updated. In such cases, the response body contains the corresponding error messages:

XML Example:

```
# Request with invalid or missing attributes

POST /users.xml

<user>

  <login>john</login>

  <lastname>Smith</lastname>
  <mail>john</mail>

</uer>



# 422 response with the error messages in its body

<errors type="array">

  <error>First name can't be blank</error>

  <error>Email is invalid</error>

</errors>
```

JSON Example:

```
# Request with invalid or missing attributes

POST /users.json

{

  "user":{

    "login":"john",

    "lastname":"Smith",

    "mail":"john"

  }

}



# 422 response with the error messages in its body

{

  "errors":[

    "First name can't be blank",

    "Email is invalid"

  ]

}
```

JSONP Support

Redmine 2.1.0+ API supports JSONP to request data from a Redmine server in a different domain (say, with JQuery). The callback can be passed using the `callback` or `jsonp` parameter. As of Redmine 2.3.0, JSONP support is optional and disabled by default, you can enable it by checking **Enable JSONP support** in Administration -> Settings -> API.

Example:

```
GET /issues.json?callback=myHandler



myHandler({"issues":[ ... ]})
```

API Usage in various languages/tools

- Ruby
- PHP
- Python
- Perl
- Java
- cURL
- Drupal Redmine API module, 2.x branch
- .NET
- Delphi

API Change history

This section lists changes to the existing API features that may have broken backward compatibility. New features of the API are listed in the API Description.

2012-01-29: Multiselect custom fields (r8721, 1.4.0)

Custom fields with multiple values are now supported in Redmine and may be found in API responses. These custom fields have a `multiple=true attribute` and their `value` attribute is an array.

Example:

```
GET /issues/296.json



{"issue":

  {

    "id":8471,

    ...

    "custom_fields":

      [

        {"value":["1.0.1","1.0.2"],"multiple":true,"name":"Affected
version","id":1},

        {"value":"Fixed","name":"Resolution","id":2}

      ]

  }
}
```