IPC Sports Data Management System (SDMS) – API v2.0

Guide to connect to the SDMS API

1 July 2019

Alexander Picolin, Head of Sports IT
## Contents

1 Objective .......................................................................................................................... 4  
2 The OAuth2 Framework ................................................................................................. 5  
3 SDMS API and OAuth2 ...................................................................................................... 6  
  3.1 Register client registration in SDMS ............................................................................ 6  
  3.2 Authorization Code Grant Type ................................................................................... 9  
  3.3 Resource Owner Password Credentials / User Credentials Grant Type ............... 11  
  3.4 Refresh Token .......................................................................................................... 12  
  3.5 Authentication Errors ............................................................................................... 14  
4 Data Requests ................................................................................................................ 16  
  4.1 Request Mechanism ................................................................................................. 16  
  4.2 Getting Resource Collection .................................................................................... 17  
  4.3 Getting Individual Resource ................................................................................... 18  
  4.4 Updating Resource ................................................................................................... 18  
  4.5 Creating Resource .................................................................................................... 19  
  4.6 Deleting Resource .................................................................................................... 19  
  4.7 HEAD ....................................................................................................................... 20  
  4.8 OPTIONS .................................................................................................................. 20  
5 Resource Endpoints ....................................................................................................... 21  
  5.1 Getting Athletes ....................................................................................................... 21  
  5.2 Updating Athlete ...................................................................................................... 26  
  5.3 Creating Athlete ....................................................................................................... 27  
  5.4 Register athlete or guide in new sport ....................................................................... 29  
  5.5 Getting Photo, Passport, Nationality Confirmation Letter .................................... 30  
  5.6 Uploading a new personal file ................................................................................... 32  
  5.7 Getting Classification History .................................................................................. 33  
  5.8 Getting Classification Profile .................................................................................... 36  
  5.9 Getting Classification Files ........................................................................................ 38
5.10 Uploading a new medical document ................................................................. 40
5.11 Resource Endpoint Errors .............................................................................. 41
6 Contact .................................................................................................................. 43
1 Objective

This document provides an overview of the possibilities offered by the International Paralympic Committee (IPC) Sports IT department to directly access the application level of the IPC Sports Data Management System (SDMS).

SDMS is used by the IPC Sports to manage athletes, their classification, licences, competitions, results, records, and competition entries, with external stakeholders, primarily the NPCs, able and obliged to access for the daily data maintenance. NPCs register and license their own athletes and keep their profiles up-to-date, while the respective IPC sport updates their classification, results, and records. Modules and further applications retrieve data from SDMS such as the IPC Online Entry System or the IPC Accreditation System to minimize data redundancy. The IPC sports administer all business processes and related data in and around SDMS. Further stakeholders like the IPC sport classifiers may or are obliged to access SDMS to perform their duties.

A common web interface exists that allows authenticated users to manage SDMS data wherever a stable Internet connection is available.

With the first developments of national sport data management versions either directly provided by the IPC or developed by a local contractor for a particular NPC, it becomes necessary that the IPC provides an application programming interface (API) that allows basic operations with athlete data through basic HTTP requests. Getting personal data, latest classification information, or recent results is one major part, but the interface also always for updates of data to some extent.

The first version of the SDMS API implemented in 2017 fulfilled the goal to transfer data between the national SDMS versions and the IPC SDMS. It was derived from but did not actually follow a standardised authorization framework. The new SDMS API version in place now makes use of the OAuth 2.0 Authorization Framework.¹

The following chapters are aimed for technical personnel that are in charge to develop a national database or similar platform that can or should establish a direct connection to SDMS for a well-defined bilateral communication and transfer of information.

¹ See https://tools.ietf.org/html/rfc6749
2 The OAuth2 Framework

The goal of the framework is to enable a third-party application (also known as 'client') to obtain limited access to an HTTP service such as SDMS. The access could be requested either on behalf of the resource owner (e.g. the NPC that had registered an athlete in SDMS) or by the client itself (e.g. the IPC website to obtain some overall data and statistics from SDMS).

In its core functionality, the client must be first authorized through one of the four standard grant types defined by the OAuth2 framework to obtain an access token. This token combines several information about the requesting and authorized client (and its associated resource owner, depending on the grant type), eventually allowing the client to finally receive data from the application (e.g. SDMS).

Some persons might be already familiar with the actual processes without knowing the underlying technical procedures. For instance, several web or mobile application (again, these are the clients) can connect to someone's own Facebook profile (the application). But the Facebook profile (resource) owner must first grant this client app access to their profile on Facebook, either by registering the client in Facebook or entering username and password in the highly trusted client app.

Four grant types are defined by this framework:

- **Resource owner Password Credentials Grant (a.k.a. User Credentials):** The user enters their application (e.g. SDMS) username and password in the client application. That requires that the user highly trusts the client to either securely store the credentials or not to store them at all but just request the access token once.
- **Client Credentials:** This grant type does not use any user login information of the application. Instead, this grant type is rather used for machine-to-machine communications, in particular if the client and application is basically the same or some other arrangement has been made but ignoring any human interference.
- **Authorization Code:** This is the most secured method and should be used when the client is a web-based application. When the client requests data from the application, the user is redirected to the application itself (e.g. SDMS), enters username and password there and confirms that they grant access of the client to SDMS. Thereafter, the user is redirected to the client application.
- **Implicit:** When it comes to script-based (e.g. JavaScript) public clients, this grant type is of use.

In addition, some of the grant types provide refresh tokens that enables the client to obtain a new access token when the previous expired without going through the authorization process again.
3 SDMS API and OAuth2

The SDMS API currently sets the only focus on the NPCs as resource owner of their own athletes registered and edited in SDMS. In order to provide maximum security, NPCs can only use the following grant types:

- Authorization Code
- Resource Owner Password Credentials / User Credentials
- Refresh Token (in combination with one of the above)

The implicit and client credentials grant types might be activated for internal use at a later stage.

The developer of a client version that shall submit to or read athlete information from SDMS must first implement the authorization procedure with any of the enabled grant types. It is highly recommended to use the Authorization Code grant type to avoid exposure of SDMS user account credentials.

3.1 Register client registration in SDMS

First step is the registration of the client in SDMS by the dedicated staff member of the NPC that has been granted access to SDMS.

1. Login to SDMS on https://db.ipc-services.org/sdms.
2. In the primary navigation bar, find the "SDMS API" icon to follow. If the icon is not there, contact sdmsadmin@paralympic.org to check for your SDMS privileges.
3. Go to 'Your Applications' and click 'Register New Application'.
4. A form appears. Enter a name for the external application (client), a short description, and if required, the Redirect URI for the authorization code grant type (see chapter XX). Check which grant type (User Credentials and/or Authorization Code) the client shall use to authorize. Finally, check whether the client may only read athlete data or also update existing and create new athletes (the scopes).
5. When saved, a summary of the submitted information will be shown, including a unique client ID representing a client secret. These values must be secretly handled like username and password. Share it only with the developer of the client application.

If the client secret has been accidentally shared or exposed, click 'Reset Client Secret' to generate a new one. That also requires that the developer of the client application must be informed about the new secret, otherwise further authorization requests will fail.
Figure 1: The registration interface for a new client

<table>
<thead>
<tr>
<th>Application Name</th>
<th>My National Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>This application represents the national database in my country to load and submit the data of our internationally competing athletes in the World Para Sports.</td>
</tr>
<tr>
<td>Redirect URI</td>
<td><a href="https://www.npc.org/app/callback">https://www.npc.org/app/callback</a></td>
</tr>
<tr>
<td>Grant Types</td>
<td>User Credentials, Authorization Code, Refresh Token</td>
</tr>
<tr>
<td>Scopes</td>
<td>Update or create athletes, Read athlete information</td>
</tr>
<tr>
<td>Client ID</td>
<td>f8d1bd94f5769ef4707</td>
</tr>
<tr>
<td>Client Secret</td>
<td>eda06ccfd401cfdf2b302c0f2ef1015f1c73d0e</td>
</tr>
<tr>
<td>Redirect URI</td>
<td><a href="https://www.npc.org/app/callback">https://www.npc.org/app/callback</a></td>
</tr>
<tr>
<td>Grant Types</td>
<td>authorization_code, refresh_token</td>
</tr>
<tr>
<td>Scope</td>
<td>athletes_edit, athletes_read</td>
</tr>
<tr>
<td>User Tokens</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 2: The summary page of the registered client
The NPC user in SDMS can change anytime the registered information for this client by clicking the 'Edit' button. This in particular allows the user to lower or raise the privileges (scopes) for the client as well as change the enabled grant types or redirect URI. If the client does not exist anymore or does not further sends requests to SDMS, it shall be deleted.

When clients are authorized, access tokens are generated. These tokens can be finally used to requests resources from SDMS. The number of existing access tokens is reflected in the summary page (User Tokens). All access tokens associated with that client can be immediately deleted by clicking the 'Revoke all User Tokens'.

All registered clients are accessible to this particular SDMS user in the 'Your Applications' tab when first opening the SDMS API section. The table reflects the name, description and scopes of the registered clients, and can be viewed, edited, and deleted from here.

The tabs of 'Endpoints', 'Authentication' and 'Authorization' provides a short summary of the available resource endpoints, authentication and authorization URIs from this document and are aimed for the IT staff of the NPC or the developers of NPC-based application.

The new both sub-chapters are intended for IT support staff to implement the authorization, authentication and resource endpoints accordingly.

It is recommended to use the Authorization Code Grant Type wherever possible.
3.2 Authorization Code Grant Type

**Step 1:** Redirect the user to this URL

https://db.ipc-services.org/sdms/api/authorize

This accepts the following GET parameters:

- `response_type` (required) must be set to `code`
- `client_id` (required) the client ID from the registration in SDMS (see Figure 2)
- `redirect_uri` (optional) the URI SDMS shall redirect the user after they granted the client the access. If the redirect URI was defined in SDMS for this client, the value must match. If the redirect URI is not provided as GET parameter, it must have been defined in SDMS for this client.
- `scope` (optional) space-separated list of scopes. SDMS API currently accepts the scopes `athletes_read` and `athletes_update`. If not set, all scopes as registered for this client in SDMS will be used.
- `state` (required) key that is returned on the redirect URI and can be used as CSRF token

The request will eventually look like

GET /sdms/api/authorize?response_type=code&client_id=f8d1bd945a765efa4707&redirect_uri=https://your.npc.site/app/callback&scope=athletes_read&state=330829d22d96 HTTP/1.1
HOST: db.ipc-services.org

When the user is not already logged in, the SDMS login page will appear. After the user logged in, another page opens, summarising that the client is asking for permission to access the available data in SDMS, respecting the provided scopes.
If the user confirms and grants permission, the user is redirected to the provided or stored redirect URI with the two GET parameters:

GET /app/callback?code=55f472608e48bb5ea0b2e26a366bec7f81e9bb7b&state=330829d22d96 HTTP/1.1
HOST: your.npc.site

Use the state parameter as CSRF token and the code for the second step to actually obtain an access token.

If the user denies the authorization, the user is also redirected to the provided or stored redirect URI with three GET parameters:

GET /app/callback?error=access_denied&error_description=The user denied access to your application&state=330829d22d96 HTTP/1.1
HOST: your.npc.site

In this case, the client shall inform the user the denying the permission will prevent any further requests of resources from SDMS.

**Step 2: Send a token request message to this URL**

https://db.ipc-services.org/sdms/oauth2/token

This accepts the following parameters in a JSON encoded array inside the request body:

- **grant_type** (required) must be set to `authorization_code`
- **code** (required) the authorization code from the callback in step 1
- **redirect_uri** (optional) must be set if the redirect URI was set in the original redirect URL in step 1 and must match the original value. Only if the redirect URI was not provided in step 1 (instead, the redirect URI was registered with the client in SDMS), this parameter must be omitted.

The client ID and the client secret must be provided as Authorization header. The full request could look like this

POST /sdms/oauth2/token HTTP/1.1
HOST: db.ipc-services.org
content-type: application/json
authorization: Basic Yjg2ZjY3MDU4MjU5MmVkY2M0NmI6N2QyYTQ2YwZkOGMzA4MjlkMjJkOTY2NDMxNWE3OTkOGfhMGQ2Yg==
content-length: 137
When successful, the SDMS API will respond back with a JSON encoded array like

```
{
  "access_token": "88604d4477c5712a98292ce2d54804c0ccad045b",
  "expires_in": 7200,
  "token_type": "Bearer",
  "scope": "athletes_read",
  "refresh_token": "a01f28412e40b98b92facc873ff1a3402369baa4"
}
```

For use of the access token for resource requests, see chapter XX. For refreshing the access token with help of the refresh token, see chapter XX.

### 3.3 Resource Owner Password Credentials / User Credentials Grant Type

Once the client has received username and password for SDMS from the user, the client sends a token request message to

```
https://db.ipc-services.org/sdms/oauth2/token
```

This accepts the following parameters in a JSON encoded array inside the request body:

- **grant_type** (required) must be set to `password`
- **username** (required) the SDMS username
- **password** (required) the SDMS password

The client ID and the client secret must be provided as Authorization header. The full request could look like this

```
POST /sdms/oauth2/token HTTP/1.1
HOST: db.ipc-services.org
content-type: application/json
authorization: Basic Yjg2ZjY3MDU4MjU5MmVkY2M0NmI6N2QyYTQ2YWZhOGMzA4MjlkMjJkOTY2NDMx
NWE3OTkOGFhMGQ2Yg==
```
When successful, the SDMS API will respond back with a JSON encoded array like

```
{
    "grant_type": "password",
    "username": "sdms_username",
    "password": "sdms_password"
}
```

By default, those scopes that are registered with the client in SDMS are returned, see Figure 1.

This grant type requires that the client application asks the resource owner for the username and password. That obviously requires high trust of the user in the client application to share SDMS username and password so that the client can request resources from SDMS on behalf of that user. Consequently, that grant type should be avoided where possible, and the authorization code grant type (see chapter 3.2) used instead.

For use of the access token for resource requests, see chapter XX. For refreshing the access token with help of the refresh token, see chapter XX.

### 3.4 Refresh Token

Access tokens are only valid for a period of two hours. Thereafter, the access token cannot be further used. Either the authorization process must be repeated or, alternatively, the client may use the refresh token that was delivered through the Authorization Code or User Credentials grant type. Refresh tokens expire after two weeks of their generation.

In order to obtain a new access token with help of a stored refresh token, the client sends a token request message to

```
https://db.ipc-services.org/sdms/oauth2/token
```
This accepts the following parameters in a JSON encoded array inside the request body:

- **grant_type** (required) must be set to **refresh_token**
- **refresh_token** (required) the refresh token from the previous authorization process

The client ID and the client secret must be provided as Authorization header. The full request could look like this:

```
POST /sdms/oauth2/token HTTP/1.1
HOST: db.ipc-services.org
content-type: application/json
authorization: Basic Yjg2ZjY3MDU4MjU5MmVkY2M0NmI6N2QyYTQ2YWZkOGMzMzA4MjlkMjJkOTY2NDMxNWE3OTkOGhMGQ2Yg==
content-length: 89

{
    "grant_type":"refresh_token",
    "refresh_token":"a01f28412e40b98b92facc873ff1a3402369bba4"
}
```

When successful, the SDMS API will respond back with a JSON encoded array like

```
{
    "access_token": "88604d4477c5712a98292ce2d54804c0ccad045b",
    "expires_in": 7200,
    "token_type": "Bearer",
    "scope": "athletes_edit athletes_read",
    "refresh_token": "4999d0c25e81f696044732d86ebd8e3eb42b704d"
}
```

The scopes are repeated from the original authorization request. If the scopes shall change, a new authorization request must be made instead using refresh tokens.
3.5 Authentication Errors

The SDMS API will return an error message when formal requirements are not met or any client ID, client secret, username, password, authorization code, refresh or access token is invalid.

When sending defective token requests to /sdms/oauth2/token, error messages will look like

```json
{
    "name": "Unauthorized",
    "message": "Invalid username and password combination",
    "code": 0,
    "status": 401
}
```

with

- **name** the HTTP status message
- **message** a descriptive message why this error occurred
- **code** an internal code (usually 0)
- **status** the HTTP status code

An overview with the error codes that might occur during the authorization process to obtain an access token and the potential reasons behind each can be found in the next table:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Status</th>
<th>Potential Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>Bad Request</td>
<td>The request was formally not correct:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Client credentials are not set in the header or invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o The grant type was not specified in the request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Grant type is not supported.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Grant type is not valid or unauthorized for the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Required request parameters could not be found or are invalid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Requested scope is invalid for the request or the client and not supported at all.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>o Provided refresh token or authorization code expired.</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>When using the Resource Owner Password Credentials grant type, the combination of username and password to authenticate the user failed. Correct username and password must be provided by the resource owner.</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
<td>The request method was not POST when requesting an access token.</td>
</tr>
</tbody>
</table>
When sending defective request to obtain an authorization code, the resource owner will still first login. Then the client ID, response type and scopes are controlled similar for token requests. If the validation fails, the resource owner will be informed by an error message similar to above.

```json
{
  "name": "Bad Request",
  "message": "The client id supplied is invalid",
  "code": 0,
  "status": 400
}
```

Figure 5: Authorization request with failed validation of client ID

The resource owner shall inform the developer of the client application and report the error that appears in SDMS. In most cases, it is an invalid client ID. It must be noted that the client ID validation might also fail if registered under another SDMS user account than the one the resource owner just logged in with.
4 Data Requests

4.1 Request Mechanism

Once an access token has been retrieved, the client may send resource requests to get or submit data for specific SDMS entities, depending on the entitlements of the registered client in SDMS and the associated SDMS user account.

The access token must be sent as authorization bearer in the request header in each request. For testing purposes, the SDMS API offers the possibility to get information about the sport names and IPC sport codes registered in SDMS. To obtain the name of the sport with IPC code 'AS', send the following request:

GET /sdms/api/sport/AS HTTP/1.1
HOST: db.ipc-services.org
content-type: application/json
authorization: Bearer 88604d4477c5712a98292ce2d54804c0ccad045b

The response will be a JSON encoded string with resource information and HATEOAS\(^2\) links.

```json
{
    "code": "AS",
    "name": "Alpine Skiing",
    "season": "Winter",
    "_links": {
        "self": {
            "href": "http://localhost/sdms/api/sport/AS"
        },
        "index": {
            "href": "http://localhost/sdms/api/sport"
        }
    }
}
```

Code, name, and season are SDMS data fields. The links provided support the client to find further information, e.g. how to list all sports registered in SDMS via the 'index' link.

---

\(^2\) Hypermedia as the Engine of Application State
In general, the resource endpoints have the following format

```
https://db.ipc-services.org/sdms/api/{resource type}[/id]
```

In the example above, resource type is 'sport'. The need for an ID of a particular resource depends on the request type, e.g. getting or updating an individual resource. The action what shall happen with a particular resource is determined by the HTTP request method. In the following list, all potentially available methods and API URIs are shown:

Which fields and which links are available depend on the resource and are explained in detail for each resource endpoint further below.

### 4.2 Getting Resource Collection

For some resources, it is allowed to obtain the full collection of all resources available, e.g. a full list of all sports registered in SDMS and can be obtained by sending a GET request to the resource endpoint like following:

```
GET /sdms/api/{resource type}[/page=n]
GET /sdms/api/sport[/page=n]
```

The returned JSON string is an array of all available resource endpoints. However, the list is limited to 20 elements at once. More data can be retrieved via the pagination mechanism of the API. Sending the above request to get all sports will send the JSON string with 20 sports, but the response header contains information for effective pagination:

```
x-pagination-total-count: 50
x-pagination-page-count: 3
x-pagination-current-page: 1
x-pagination-per-page: 20
link: <https://db.ipc-services.org/sdms/api/sport?page=1>; rel=self, <https://db.ipc-services.org /sdms/api/sport?page=2>; rel=next, <https://db.ipc-services.org /sdms/api/sport?page=3>; rel=last
```

The x-pagination entries represent the total number of resources in the collection (e.g. 50), the number of 'pages' to collect all resources (3), the current page (1), and the number of resources per page (20).

The link entry helps to find the right ways how to request the different pages. Depending on the current page and the number of pages available, you will find:
• `rel=self`: the link to the currently requested page
• `rel=first`: the link to the first page
• `rel=prev`: the link to the previous page
• `rel=next`: the link to the next available page
• `rel=last`: the link to the last page

Obviously, you can request any page by sending

```plaintext
GET /sdms/api/{resource type}?page={n}
```

With `{n}` representing a page between 1 and the value of `x-pagination-page-count`. Default page value is 1 and can be omitted. To get the second page of the sports collection, send

```plaintext
GET /sdms/api/sport?page=2
```

### 4.3 Getting Individual Resource

To obtain data for a specific resource, just send

```plaintext
GET /sdms/api/{resource type}/{resource id}
```

GET /sdms/api/sport/AS

The returned JSON string is an object with the accessible resource attributes and some useful links to obtain further related information of that resource or the entire collection. Compare the example in chapter 4.1.

### 4.4 Updating Resource

For individual resources and specific attributes, an update request can be sent, using the PATCH or PUT method.

```plaintext
PUT /sdms/api/{resource type}/{resource id}
PATCH /sdms/api/{resource type}/{resource id}
```

The body of the request must contain the values to be changed. The array keys represent the attribute name. Not all fields that are shown after a GET request can be edited through the API.
In particular, the identifier for a resource is not editable, e.g. the SDMS ID of an athlete. Which fields are editable will be defined for each resource in the next main chapter.

SDMS API will also validate the data entries. On validation errors, the JSON string represents an array of all error messages. An error could look like this:

```
[
  {
    "field": "name",
    "message": "Name cannot be blank."
  }
]
```

The client must correct all errors in order to successfully submit changes to a resource.

Once validation succeeds, the changes are applied, and the final resource data are returned like with a standard GET request on this resource.

### 4.5 Creating Resource

For individual resources, a request to create a new resource can be sent, using the POST method.

```
POST /sdms/api/{resource type}
```

The body of the request must contain all required values. The array keys represent the attribute name. Not all fields that are shown after a GET request can be created through the API, some are auto-generated by SDMS.

Like for an update of an existing resource, validation will be conducted whether all required fields are sent, and each field obey to a certain set of validation rules. On validation errors, the JSON string returned represents an array of all error message. The client must correct all errors in order to successfully register a new resource.

Once validation succeeds, the new resource will be registered, and the final resource data are returned like with a standard GET request on this newly created resource.

### 4.6 Deleting Resource

Finally, an existing resource could be deleted. The SDMS API however will not perform any delete operations, unless the logic of SDMS requires that internal when existing resources are updated, or new resources created. Generally, a delete request would look like
DELETE /sdms/api/{resource type}/{resource id}

That would irrevocably delete a resource of the specified resource type with that particular ID from SDMS.

4.7 HEAD

It is possible to send a HEAD request to a resource collection or an individual resource.

HEAD /sdms/api/{resource type}
HEAD /sdms/api/{resource type}/{resource id}
HEAD /sdms/api/sport
HEAD /sdms/api/sport/AS

The exact same header information will be returned (e.g. pagination and links for a resource collection) as described in 4.2 and 4.3. Only difference is that no resource information will be available in the response body as per definition of the HEAD method.³

4.8 OPTIONS

The OPTIONS method allows to obtain a list of all allowed HTTP methods for the given resource endpoint.

OPTIONS /sdms/api/{resource type}
OPTIONS /sdms/api/{resource type}/{resource id}

The list can be obtained from the response header 'allow', e.g.

Allow: GET, PUT, PATCH, HEAD, OPTIONS

³ See https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html
5 Resource Endpoints

The following chapter considers all endpoints available in SDMS. It must be repeated that the user associated with the client ID must have the related SDMS privileges and the right scope in the associated access token to access the endpoints.

5.1 Getting Athletes

Resource collection (see chapter 4.2):

GET /sdms/api/athlete

Individual resource identified by their SDMS ID {id}:

GET /sdms/api/athlete/{id}

Response example (individual resource, otherwise array of objects like below):

```json
{
    "id": 1000,
    "type": "ATH",
    "npc": "GBR",
    "family-name": "Smith",
    "given-name": "John Adam",
    "preferred-family-name": "Smith",
    "preferred-given-name": "John",
    "gender": "Male",
    "date-of-birth": "1977-07-07",
    "career-status": "ACT",
    "nationality": "GBR",
    "passport-number": "ABCDEF123456",
    "passport-type": "PAS",
    "passport-expiry": "2020-07-23",
    "previous-npc": null,
    "intellectual-impairment": "N",
    "inas-id": null,
    "last-modification": 1503574824,
    "_links": {
        "self": {
            "href": "https://db.ipc-services.org/sdms/api/athlete/1000"
        },
        "index": {
         ...}
```
Attribute description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Editable</th>
<th>On Create</th>
<th>SDMS Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>No</td>
<td>Auto</td>
<td>SDMS ID</td>
<td>Unique athlete ID, integer; automatically assigned when new athlete is registered</td>
</tr>
<tr>
<td>type</td>
<td>No</td>
<td>Mandatory</td>
<td>Participant Type</td>
<td>&quot;ATH&quot; for athlete or &quot;GUI&quot; for guide</td>
</tr>
<tr>
<td>npc</td>
<td>No</td>
<td>Mandatory</td>
<td>NPC</td>
<td>Official three-letter NPC code</td>
</tr>
<tr>
<td>family-name</td>
<td>Yes*</td>
<td>Mandatory</td>
<td>Family Name (passport)</td>
<td>Family name of athlete according to recent passport. Can be empty/null.</td>
</tr>
<tr>
<td>given-name</td>
<td>Yes*</td>
<td>Mandatory</td>
<td>Given Name (passport)</td>
<td>Given name of athlete according to recent passport.</td>
</tr>
<tr>
<td>preferred-family-name</td>
<td>Yes</td>
<td>Optional</td>
<td>Family Name (preferred)</td>
<td>Preferred family name of athlete</td>
</tr>
<tr>
<td>preferred-given-name</td>
<td>Yes</td>
<td>Optional</td>
<td>Given Name (preferred)</td>
<td>Preferred given name of athlete</td>
</tr>
<tr>
<td>gender</td>
<td>No</td>
<td>Mandatory</td>
<td>Gender</td>
<td>&quot;Male&quot; or &quot;Female&quot;</td>
</tr>
<tr>
<td>date-of-birth</td>
<td>No</td>
<td>Mandatory</td>
<td>Date of Birth</td>
<td>Format yyyy-mm-dd</td>
</tr>
<tr>
<td>career-status</td>
<td>No</td>
<td>Default</td>
<td>Career Status</td>
<td>“ACT” for active or “RET” for retired. New athletes are automatically considered as active.</td>
</tr>
<tr>
<td>nationality</td>
<td>Yes</td>
<td>Mandatory</td>
<td>Nationality</td>
<td>ISO 3166-1 Code (not the NPC/NOC code!) of athlete’s nationality.</td>
</tr>
<tr>
<td>passport-number</td>
<td>Yes</td>
<td>Mandatory</td>
<td>Passport/ ID Card No</td>
<td>Number of nationality document</td>
</tr>
<tr>
<td>passport-type</td>
<td>Yes</td>
<td>Mandatory</td>
<td>Validation Document</td>
<td>“PAS” for passport or “IDC” for national ID cards</td>
</tr>
<tr>
<td>passport-expiry</td>
<td>Yes</td>
<td>Mandatory</td>
<td>Date of Expiration</td>
<td>When the document expires; is null if the document never expires.</td>
</tr>
<tr>
<td>previous-npc</td>
<td>Yes</td>
<td>Optional</td>
<td>Has the athlete ever competed for another NPC before?</td>
<td>Official three-letter NPC code, null if none.</td>
</tr>
<tr>
<td>Field</td>
<td>Required</td>
<td>Optional</td>
<td>Description</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>intellectual-impairment</td>
<td>Yes</td>
<td>Optional</td>
<td>Intellectual Impairment</td>
<td>Whether the athlete has an intellectual impairment (Y/N)</td>
</tr>
<tr>
<td>inas-id</td>
<td>Yes</td>
<td>Optional</td>
<td>INAS Classification No</td>
<td>For intellectually impaired athletes, the INAS classification number officially registered. Empty string or null if no INAS ID is registered.</td>
</tr>
<tr>
<td>last-modification</td>
<td>No</td>
<td>Default</td>
<td>-</td>
<td>UNIX timestamp of last modification.</td>
</tr>
</tbody>
</table>

Additionally, related information to the athlete can be included in the response message. To retrieve one or more related data objects inside the athlete object, the request has to be adjusted like follows for a resource collection:

GET /sdms/api/athlete?expand={relation1}[,{relation2}[,...]]

or for an individual resource identified by their SDMS ID {id}:

GET /sdms/api/athlete/{id}?expand={relation1}[,{relation2}[,...]]

with relationX being one of the accessible relations:

<table>
<thead>
<tr>
<th>Relation</th>
<th>Description</th>
<th>see chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>photo</td>
<td>Athlete’s photo including photo file content (base64 encoded), photo meta data (dimensions and size) and hash value</td>
<td>5.2</td>
</tr>
<tr>
<td>photoHash</td>
<td>Same as photo, but without file content</td>
<td>5.2</td>
</tr>
<tr>
<td>passport</td>
<td>Athlete’s current passport including file content (base64 encoded), file size, file type and hash value</td>
<td>5.2</td>
</tr>
<tr>
<td>passportHash</td>
<td>Same as passport, but without file content</td>
<td>5.2</td>
</tr>
<tr>
<td>nationalityLetter</td>
<td>IPC nationality confirmation letter including file content (base64 encoded), file size, file type and hash value</td>
<td>5.2</td>
</tr>
<tr>
<td>nationalityLetterHash</td>
<td>Same as nationalityLetter, but without file content</td>
<td>5.2</td>
</tr>
<tr>
<td>classification</td>
<td>Array of current and historical classification</td>
<td>5.3</td>
</tr>
<tr>
<td>profile</td>
<td>Array of current classification profile information</td>
<td>5.4</td>
</tr>
<tr>
<td>classFiles</td>
<td>Array of current classification files (class sheets and medical documentation) including file content (base64 encoded), file size, file type and hash value</td>
<td>5.5</td>
</tr>
<tr>
<td>classFilesHash</td>
<td>Same as classFiles, but without file content</td>
<td>5.5</td>
</tr>
</tbody>
</table>
Following request would provide an individual athlete resource including photo and classification:

GET /sdms/api/athlete/1000?expand=photo,classification

Response example (shortened):

```
{
  "id": 1000,
  "npc": "GBR",
  [...]
  "last-modification": 1503574824,
  "photo": {
    "id": 1000,
    "size": 56321,
    "width": 176,
    "height": 198,
    "mime-type": "image/jpeg",
    "hash": "cb9b4d3cd2e19b41939b59c4fbb7efef",
    "last-modification": 1561539675,
    "data": "/9j/4AAQSkZJ [...]", // string shortened for readability
    "_links": [...] 
  },
  "classification": [
    {
      "sport": "AT",
      "class": "F13",
      "status": "N",
      "last-modification": 1460361537,
      "_links": [...] 
    },
    [...] // more athlete class objects
  ],
  "_links": [...] 
}
```

Attention must be paid regarding the difference between photo (single object) and classification of the athlete (array of class objects).

It is also possible to directly access a related object or an array of related objects:

GET /sdms/api/athlete/{id}/{relation}
To retrieve the photo for the athlete with the SDMS ID 1000, the request to be sent is:

GET /sdms/api/athlete/1000/photo

Further details of the related models can be found in the next chapters.

Each object comes with a meta attributes called `_links`. This attribute is an object describing the request URLs for parent or children resources. It always contains the self link, e.g. for reloading the currently retrieved object.

```json
{
  "id": 1000,
  ...
  "_links": {
    "self": {
      "href": "https://db.ipc-services.org/sdms/api/athlete/1000"
    },
    "index": {
      "href": "https://db.ipc-services.org/sdms/api/athlete"
    },
    "photo": {
      "href": "https://db.ipc-services.org/sdms/api/athlete/1000/photo"
    },
    ...
  }
}
```

The example above defines three links:

- The `self` link to reload the current athlete resource.
- The `index` link to load the resource collection. The current athlete resource will be included in that collection.
- The `photo` link to load the photo resource for that athlete.

These links allow the client to navigate through the hierarchy of data using the provided links information.
5.2 Updating Athlete

Following requests trigger an update of an existing athlete with SDMS ID \{id\}:

PATCH /sdms/api/athlete/{id}
PUT /sdms/api/athlete/{id}

In the SDMS API, the methods PATCH and PUT are aliases for each other. The request must be expanded with a JSON encoded array of the attributes to be changed, with the keys as the attributes as per table in chapter 5.1. This table identifies which attributes of the resource can be changed in the 'editable' column. Invalid attribute IDs will be ignored, meaning no error thrown.

Send the following request to update the passport number for the athlete with SDMS ID 1000.

```plaintext
PATCH /sdms/api/athlete/1000 HTTP/1.1
HOST: db.ipc-services.org
authorization: Bearer 88604d4477c5712a98292ce2d54804c0ccad045b
content-type: application/json
{
   "passport-number": "GHIJ7654321"
}
```

Each submitted value will be validated against SDMS rules like in the SDMS frontend. When the validation succeeds, the sent attributes will be updated, and the response will contain the updated athlete profile like in a GET request (5.1).

The update process accepts three more attributes that are not included in GET request.

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Editable</th>
<th>On Create</th>
<th>SDMS Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comments</td>
<td>Yes</td>
<td>Optional</td>
<td>NPC Comments</td>
<td>Specific comments for this athlete, shown in the NPC comment field. Previous value is overwritten.</td>
</tr>
<tr>
<td>name-change-reason</td>
<td>No *</td>
<td>n/a</td>
<td>Reason for athlete name change</td>
<td>Obligatory to send when passport family name or given is changed. Must be one of the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• E: Previous name was incorrect</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• M: Name change due to wedding/divorce/similar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• N: Name change after nationality change</td>
</tr>
<tr>
<td>name-change-date</td>
<td>No *</td>
<td>n/a</td>
<td>Affective from</td>
<td>Obligatory to send when passport family name or given name is changed.</td>
</tr>
</tbody>
</table>
(*) Special rules apply when the passport family or give name is changed, e.g. after a wedding of the athlete. In the SDMS frontend, the user is asked for the reason why the name changed, and since when the new name is in place. The API replicates that questionnaire. When family or given name is changed, the attributes name-change-reason and name-change-date must be sent alongside to specify why and when the name changed.

As example, the request to change the family name of athlete 1000 to Miller, affective from 27 February 2019, due to a wedding has to look like the following:

```plaintext
PATCH /sdms/api/athlete/1000 HTTP/1.1
HOST: db.ipc-services.org
authorization: Bearer 88604d4477c5712a98292ce2d54804c0ccad045b
content-type: application/json
{
    "family-name": "Miller",
    "preferred-family-name": "Miller",
    "name-change-reason": "M",
    "name-change-date": "2019-02-07"
}
```

5.3 Creating Athlete

Following request triggers the creation of a new athlete in SDMS:

```plaintext
POST /sdms/api/athlete
```

The request must be expanded with a JSON encoded array of all required attributes as per table in chapter 5.1. This table identifies which attributes of the resource are required to register in a new athlete in the 'on create' column. Invalid attribute IDs will be ignored, meaning no error thrown. In addition to the attributes in chapter 5.1, additionally attributes are accepted and partially required:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>On Create</th>
<th>SDMS Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Mandatory</td>
<td>Category</td>
<td>Whether it is an athlete (&quot;ATH&quot;) or a competition partner/guide (&quot;GUI&quot;). Guides do not apply any Para classification.</td>
</tr>
<tr>
<td>sport</td>
<td>Mandatory</td>
<td>Sport</td>
<td>The first sport to register the athlete in. Must be represented by the two-letter sport codes as defined by the IPC.</td>
</tr>
<tr>
<td>class1</td>
<td>Mandatory if ATH</td>
<td>1st Class</td>
<td>The primary class of the athlete. Must be represented by the class codes as defined by the IPC.</td>
</tr>
</tbody>
</table>
Each class is registered with the ‘New’ classification status by default.  

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>class2</td>
<td>Optional 2nd Class. The second class of the athlete for multi-class sports (e.g. athletics or swimming).</td>
</tr>
<tr>
<td>class3</td>
<td>Optional 3rd Class. The third class of the athlete.</td>
</tr>
</tbody>
</table>

Full request with the minimum number of fields to register a new athlete could be:

```
POST /sdms/api/athlete HTTP/1.1
HOST: db.ipc-services.org
authorization: Bearer 88604d4477c5712a98292ce2d54804c0ccad045b
content-type: application/json
{
    "type": "ATH",
    "npc": "USA",
    "given-name": "Jane",
    "family-name": "Johnson",
    "gender": "Female",
    "date-of-birth": "1998-04-23",
    "nationality": "USA",
    "passport-type": "PAS",
    "passport-number": "ZYX987 654 321",
    "passport-expiry": "2024-12-12",
    "sport": "AT",
    "class1": "T42"
}
```

When validation of the submitted data succeeds, the athlete is registered with a new ID. The response to the request includes the full athlete profile as per GET request on the number ID. It is advised to use the newly generated SDMS ID and store the number in the client’s database for further synchronisation or update requests.

Attention must be paid to the fact that few attributes can only be set once when creating a new athlete but not edited at a later stage, e.g. gender or date of birth.

---

4 For all class codes, see [https://db.ipc-services.org/centre/common-codes/index/entity/class](https://db.ipc-services.org/centre/common-codes/index/entity/class)

5 Only non-pseudo classes and classes eligible for first registration (depending on sport regulations) are eligible to be selected.
5.4 Register athlete or guide in new sport

Following request triggers the registration of an existing athlete with SDMS ID {id} in a new sport:

POST /sdms/api/athlete/{id}/classification

The request must be expanded with a JSON encoded array with following attributes:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>On Create</th>
<th>SDMS Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sport</td>
<td>Mandatory</td>
<td>Sport</td>
<td>Sport code (only for World Para Sports)</td>
</tr>
<tr>
<td>class1</td>
<td>Mandatory if athlete</td>
<td>1st Class</td>
<td>The primary class of the athlete. Must be represented by the class codes as defined by the IPC. Each class is registered with the ‘New’ classification status by default.</td>
</tr>
<tr>
<td>class2</td>
<td>Optional</td>
<td>2nd Class</td>
<td>The second class of the athlete for multi-class sports (e.g. athletics or swimming).</td>
</tr>
<tr>
<td>class3</td>
<td>Optional</td>
<td>3rd class</td>
<td>The third class of the athlete.</td>
</tr>
</tbody>
</table>

Athletes can only be registered in a sport which they are not classified in. It is possible to add a second or third class at later stage as long as the athlete has not been classified, yet. Guides can only be registered in a new sport if not already registered in.

Full request to register an existing athlete with SDMS ID 1000 in swimming could be:

POST /sdms/api/athlete/1000/classification HTTP/1.1
HOST: db.ipc-services.org
authorization: Bearer 88604d4477c5712a98292ce2d54804c0ccad045b
content-type: application/json

{
    "sport": "SW",
    "class1": "S08",
    "class2": "SB07",
    "class3": "SM08",
}

Only non-pseudo classes and classes eligible for first registration (depending on sport regulations) are eligible to be selected.

When validation of the submitted data succeeds, the new sport or new classes are registered for the athlete. The response to the request returns an array of all newly registered class objects.

---

6 For all class codes, see [https://db.ipc-services.org/centre/common-codes/index/entity/class](https://db.ipc-services.org/centre/common-codes/index/entity/class)
5.5 Getting Photo, Passport, Nationality Confirmation Letter

Athlete resource collection including the photo, passport or nationality confirmation letter as attribute (with and without file content):

GET /sdms/api/athlete?expand=photo
GET /sdms/api/athlete?expand=photoHash
GET /sdms/api/athlete?expand=passport
GET /sdms/api/athlete?expand=passportHash
GET /sdms/api/athlete?expand=nationalityLetter
GET /sdms/api/athlete?expand=nationalityLetterHash

Athlete resource with SDMS ID {id} including the photo as attribute (with and without file content):

GET /sdms/api/athlete/{id}?expand=photo
GET /sdms/api/athlete/{id}?expand=photoHash
GET /sdms/api/athlete/{id}?expand=passport
GET /sdms/api/athlete/{id}?expand=passportHash
GET /sdms/api/athlete/{id}?expand=nationalityLetter
GET /sdms/api/athlete/{id}?expand=nationalityLetterHash

Athlete photo resource (with and without file content) for athlete with SDMS ID {id}:

GET /sdms/api/athlete/{id}/photo
GET /sdms/api/athlete/{id}/photoHash
GET /sdms/api/athlete/{id}/passport
GET /sdms/api/athlete/{id}/passportHash
GET /sdms/api/athlete/{id}/nationalityLetter
GET /sdms/api/athlete/{id}/nationalityLetterHash

Response example (using the direct relationship /photo):

```json
{
    "athlete": 1000,
    "size": 56321,
    "width": 176,
    "height": 198,
    "mime-type": "image/jpeg",
    "hash": "cb9b4d3cd2e19b41939b59c4fbb7efef",
    "last-modification": 1561539675,
}
```
"data": "/9j/4AAQSkZJ...", // excluded if relationship = photoHash
"_links": {
  "self": {
    "href": "https://db.ipc-services.org/sdms/api/athlete/1000/photo"
  },
  "hash-only": {
    "href": "https://db.ipc-services.org/sdms/api/athlete/1000/photoHash"
  },
  "athlete": {
    "href": "https://db.ipc-services.org/sdms/api/athlete/1000"
  }
}

Attribute description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>athlete</td>
<td>Integer</td>
<td>SDMS ID of athlete</td>
</tr>
<tr>
<td>size</td>
<td>Integer</td>
<td>Size of file in bytes</td>
</tr>
<tr>
<td>width</td>
<td>Integer</td>
<td>Width of image in pixels (for photos only)</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>Height of image in pixels (for photos only)</td>
</tr>
<tr>
<td>mime-type</td>
<td>String</td>
<td>Mime type of file. For photos, it always returns &quot;image/jpeg&quot; (photos are auto-converted to JPG during upload to SDMS) and for all other files &quot;application/pdf&quot; (might be extended in future).</td>
</tr>
<tr>
<td>data</td>
<td>Base64-encoded string</td>
<td>The file content. The binary content is base64-encoded so that the requestor needs to decode it for final storage. This attribute is excluded for the hash version of the relationship.</td>
</tr>
<tr>
<td>hash</td>
<td>String</td>
<td>MD5-hash of the original file content. Only sent if the request includes the 'hash' parameter.</td>
</tr>
<tr>
<td>last-modification</td>
<td>Integer</td>
<td>UNIX timestamp of last modification.</td>
</tr>
</tbody>
</table>
5.6 Uploading a new personal file

Send one of the following requests to send a new photo or passport copy for athlete with SDMS ID \{id\}:

POST /sdms/api/athlete/{id}/photo
POST /sdms/api/athlete/{id}/passport

The request must be expanded with a JSON encoded array with exactly one attribute:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>On Create</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Mandatory</td>
<td>The base64-encoded content of the file. Photos must be a valid jpg, png, or gif image not larger than 1 MB (before encoding to base64) in portrait format (image height &gt; image width). Photos are automatically converted to JPG and resized to a maximum of 176x256 pixels. Passport copies must be sent as PDF file, not larger than 1.25 MB.</td>
</tr>
</tbody>
</table>

Please note that only the POST method is a valid command. In general, the uploaded file is always considered as a new file. Uploading a photo replaces a previously uploaded photo if any photo existed before. Uploading a new passport copy archives the previous one if any copy was uploaded before. Consequently, for the SDMS API these files are always considered as being ‘newly created’.

When validation of the submitted data succeeds, the file is stored on the SDMS server and linked to the athlete profile. The response to the request includes the file information as per GET request.

It is also possible to use the hashed versions of the requests, e.g.

POST /sdms/api/athlete/{id}/photoHash
POST /sdms/api/athlete/{id}/passportHash

The upload process of the file is the same, but the response excludes the data. That might be useful to keep the data transfer on a reasonable level allowing to check that the file has been correctly uploaded by comparing file hashes. The process on client’s side would be:

- Create hash of file content.
- Create base64-encoded data string of file content and send via SDMS API.
- Load the response from the SDMS API and compare hash with locally created hash.
- If hashes are the same, exit process. If different, repeat the upload.
5.7 Getting Classification History

Athlete resource collection including classification history:
GET /sdms/api/athlete?expand=classification

Athlete resource with SDMS ID {id} including classification history:
GET /sdms/api/athlete/{id}?expand=classification

Resource collection of classification history for athlete with SDMS ID {id}:
GET /sdms/api/athlete/{id}/classification

Response example (using the direct relationship /classification):

```
[
  {
    "athlete": 1000,
    "sport": "AT",
    "class": "F13",
    "status": "R",
    "last-modification": 1460361537,
    "from": "2016-04-05",
    "review": 2020,
    "classifier1": "Jane Smith",
    "classifier2": "Michael Mueller",
    "_links": {
      "self": {
        "href": "[…]/sdms/api/athlete/1000/classification"
      },
      "athlete": {
        "href": "[…]/sdms/api/athlete/1000"
      }
    }
  },
  {
    "athlete": 1000,
    "sport": "AT",
    "class": "T13",
    "status": "C",
    "last-modification": 1501327821,
    "from": "2018-12-03",
    "classifier1": "Jane Smith",
  }
]
```
"classifier2": "Michael Mueller",
"_links": [...]
},
{
  "athlete": 1000,
  "sport": "SW",
  "class": "S13",
  "status": "N",
  "last-modification": 1509358817,
  "_links": [...]
}
]

**Attribute description**

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Always Sent</th>
<th>SDMS Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>athlete</td>
<td>Yes</td>
<td>SDMS ID</td>
<td>SDMS ID of athlete</td>
</tr>
<tr>
<td>sport</td>
<td>Yes</td>
<td>Sport</td>
<td>Sport code of class</td>
</tr>
<tr>
<td>class</td>
<td>Yes</td>
<td>Class</td>
<td>Class of athlete (SDMS class code, not class name!)</td>
</tr>
<tr>
<td>status</td>
<td>Yes</td>
<td>Classification Status</td>
<td>“N” for New, “R” for Review or “C” for Confirmed</td>
</tr>
<tr>
<td>last-modification</td>
<td>Yes</td>
<td>-</td>
<td>UNIX timestamp of last modification.</td>
</tr>
<tr>
<td>from</td>
<td>No</td>
<td>Classification Date</td>
<td>Date of classification of that class. Only sent if status is “R” or “C”.</td>
</tr>
<tr>
<td>from-sort</td>
<td>No</td>
<td>Classification Date Order</td>
<td>Integer to represent the order of classification if more than one panel appeared on the same day (usually at protests). Only sent if set and status is “R” or “C”.</td>
</tr>
<tr>
<td>review</td>
<td>No</td>
<td>Year of Review</td>
<td>Four-digit year. Only sent if set and status is “R”.</td>
</tr>
<tr>
<td>reason</td>
<td>No</td>
<td>Reason for (Re)Classification</td>
<td>“BAC” (Board of Appeal of Classification), “MRR” (Medical Review Request), or “PuEC” (Protest under Exceptional Circumstances)</td>
</tr>
<tr>
<td>lim-class-group</td>
<td>No</td>
<td>Limitation (Class Group)</td>
<td>an SDMS class group identifier if class is limited to a specific class group</td>
</tr>
<tr>
<td>lim-evt-category</td>
<td>No</td>
<td>Limitation (Event Type Category)</td>
<td>an SDMS event type category identifier if class is limited to a specific event type category</td>
</tr>
<tr>
<td>lim-evt-group</td>
<td>No</td>
<td>Limitation (Event Type Group)</td>
<td>an SDMS event type group identifier if class is limited to a specific event type group</td>
</tr>
<tr>
<td>----------------</td>
<td>-----</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>classifier1</td>
<td>No</td>
<td>1st Classifier</td>
<td>Name or acronym of first classifier</td>
</tr>
<tr>
<td>classifier2</td>
<td>No</td>
<td>2nd Classifier</td>
<td>Name or acronym of second classifier</td>
</tr>
<tr>
<td>classifier3</td>
<td>No</td>
<td>3rd Classifier</td>
<td>Name or acronym of third classifier</td>
</tr>
<tr>
<td>historic</td>
<td>No</td>
<td>Current vs Historical</td>
<td>Flags with “Y” that this class belongs to the athlete’s classification history. Only sent if class is historic. Historical classes with “N” (New) status are never sent.</td>
</tr>
</tbody>
</table>
5.8 Getting Classification Profile

Athlete resource collection including classification profile:
GET /sdms/api/athlete?expand=profile

Athlete resource with SDMS ID {id} including classification profile:
GET /sdms/api/athlete/{id}?expand=profile

Resource collection of classification profile values for athlete with SDMS ID {id}:
GET /sdms/api/athlete/{id}/profile

Response example (using the direct relationship /profile):
[
  {
    "athlete": 1000,
    "sport": "AT",
    "code": "APR00DIAGNOSIS",
    "label": "Diagnosis",
    "value": "Hypertonia - Hemiplegic",
    "last-modification": 1504878849,
    "_links": {
      "self": {
        "href": "[...]/sdms/api/athlete/1000/profile"
      },
      "athlete": {
        "href": "[...]/sdms/api/athlete/1000"
      }
    }
  },
  {
    "athlete": 1000,
    "sport": "AT",
    "code": "APRATHEIGHT",
    "label": "Maximum Standing Height",
    "value": "178.5",
    "last-modification": 1504690938,
    "_links": [...]
  }
]
"sport": "SW",
"code": "APR00DIAGNOSIS",
"label": "Diagnosis",
"value": "Tetra",
"last-modification": 1504878935,
"_links": [...]
},
{
  "athlete": 1000,
  "sport": "SW",
  "code": "APRSWEXCEPT",
  "label": "Exceptions",
  "value": ["A", "12", "+"],
  "last-modification": 1504691112,
  "_links": [...]
},
]

Attribute description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Always Sent</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>athlete</td>
<td>Yes</td>
<td>SDMS ID of athlete</td>
</tr>
<tr>
<td>sport</td>
<td>Yes</td>
<td>Sport code of profile field</td>
</tr>
<tr>
<td>code</td>
<td>Yes</td>
<td>IPC SDMS internal profile code for unique identification</td>
</tr>
<tr>
<td>label</td>
<td>Yes</td>
<td>Label of the profile field</td>
</tr>
<tr>
<td>value</td>
<td>Yes</td>
<td>Value of the profile field. Can be either a string, a number (integer of float) or again an array of individual values (e.g. multi-check boxes)</td>
</tr>
<tr>
<td>last-modification</td>
<td>Yes</td>
<td>UNIX timestamp of last modification.</td>
</tr>
</tbody>
</table>
5.9 Getting Classification Files

Athlete resource collection including collection of current classification files (with and without file content):
GET /sdms/api/athlete?expand=classFiles
GET /sdms/api/athlete?expand=classFilesHash

Athlete resource with SDMS ID \{id\} including current classification files (with and without file content):
GET /sdms/api/athlete/{id}?expand=classFiles
GET /sdms/api/athlete/{id}?expand=classFilesHash

Resource collection of current classification files for athlete with SDMS ID \{id\} with and without file content:
GET /sdms/api/athlete/{id}/classFiles
GET /sdms/api/athlete/{id}/classFilesHash

Response example (using the direct relationship /classFiles):
[
    {
        "athlete": 1000,
        "sport": "SW",
        "code": "CLF00SHEET1",
        "label": "Classification Sheet 1",
        "size": 106478,
        "mime-type": "application/pdf",
        "hash": "0b476fdc94fe8c94c0153aa8b3583fbc",
        "last-modification": 1561552816,
        "data": "JVBERi0xLjUN[…]", // excluded if relationship = classFilesHash
        "_links": {
            "self": {
                "href": "[…]/sdms/api/athlete/1000/classFiles"
            },
            "hash-only": {
                "href": "[…]/sdms/api/athlete/1000/classFilesHash"
            },
            "athlete": {
                "href": "[…]/sdms/api/athlete/1000"
            }
        }
    }
]
Attribute description

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>athlete</td>
<td>Integer</td>
<td>SDMS ID of related athlete</td>
</tr>
<tr>
<td>sport</td>
<td>String</td>
<td>Sport code of classification file</td>
</tr>
<tr>
<td>code</td>
<td>String</td>
<td>IPC SDMS internal file container identifier</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Label of the file container</td>
</tr>
<tr>
<td>size</td>
<td>Integer</td>
<td>Size of file in bytes</td>
</tr>
<tr>
<td>mime-type</td>
<td>String</td>
<td>Mime type of file. Classification files are all “application/pdf” (further extensions possible in future).</td>
</tr>
<tr>
<td>hash</td>
<td>String</td>
<td>MD5-hash of the original file content.</td>
</tr>
<tr>
<td>last-modification</td>
<td>Integer</td>
<td>UNIX timestamp of last modification.</td>
</tr>
<tr>
<td>data</td>
<td>base64-encoded string</td>
<td>The file content. The binary content is base64-encoded so that the requestor needs to decode it for final storage. This attribute is excluded for the hash version of the relationship.</td>
</tr>
</tbody>
</table>
5.10 Uploading a new medical document

Send the following request to send a new medical document for athlete with SDMS ID \{id\}:

\[
\text{POST} /sdms/api/athlete/{id}/classFile
\]

The request must be expanded with a JSON encoded array with exactly three attributes:

<table>
<thead>
<tr>
<th>Attribute Name</th>
<th>On Create</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sport</td>
<td>Mandatory</td>
<td>Sport code. The athlete with ID {id} must be registered for that sport.</td>
</tr>
<tr>
<td>code</td>
<td>Mandatory</td>
<td>The code for the file container. Following codes are accepted:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CFE00MEDICAL: Medical Diagnostic Form (MDF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CFE00TSAL: TSAL Form for II athletes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CFE00MDS: Supporting Medical Documentation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• CFESHAED: Athlete Epilepsy Declaration (shooting only!)</td>
</tr>
<tr>
<td>data</td>
<td>Mandatory</td>
<td>The base64-encoded content of the file. Each file must be sent as PDF file, not larger than 5 MB. It is kindly requested to minimise file size as much as possible before transferring it without lowering file quality too much.</td>
</tr>
</tbody>
</table>

Please note that only the \text{POST} method is a valid command. When submitting a new medical document, any previous version moves to the SDMS classification file archive (accessible through the SDMS web interface only). Consequently, for the SDMS API these files are always considered as being ‘newly created’.

When validation of the submitted data succeeds, the file is stored on the SDMS server and linked to the athlete’s classification profile. The response to the request includes the file information (meaning a single classification file object in opposite to a call of \text{GET} /sdms/api/athlete/{id}/classFiles).

It is also possible to use the hashed versions of the requests, e.g.

\[
\text{POST} /sdms/api/athlete/{id}/classFileHash
\]

The upload process of the file is the same, but the response excludes the data. That might be useful to keep the data transfer on a reasonable level allowing to check that the file has been correctly uploaded by comparing file hashes. The process on client’s side would be:

- Create hash of file content.
- Create base64-encoded data string of file content and send via SDMS API.
- Load the response from the SDMS API and compare hash with locally created hash.
- If hashes are the same, exit process. If different, repeat the upload.
5.11 Resource Endpoint Errors

Instead of the expected results and resources, the SDMS API might respond with an error when the request for a response is invalid. There are two types of errors:

- The request itself was invalid (error codes 401, 403, 404, 405).
- Request is valid but validation of submitted data to update or create a new resource has failed (error code 422).
- SDMS has an error (error code 500).

An overview with the error codes that might occur during the authorization process to obtain an access token and the potential reasons behind each can be found in the next table:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Error Status</th>
<th>Potential Reasons</th>
</tr>
</thead>
<tbody>
<tr>
<td>401</td>
<td>Unauthorized</td>
<td>The access token is invalid or has expired. The client needs to request a new access token or refresh the previous one (see chapter 3)</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
<td>The requested resource endpoint is forbidden for the SDMS user associated with the provided access token.</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
<td>The requested resource could not be found. Either there is no such resource with the provided ID or the SDMS user associated with the provided access token has not the privileges to read or edit information of the resource.</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
<td>An invalid request method was used for the resource point. Please check which request methods are allowed for each endpoint.</td>
</tr>
<tr>
<td>422</td>
<td>Data Validation Failed</td>
<td>When updating or creating a resource, the provided attribute values were not valid. The array in the response message body specify the errors to be corrected.</td>
</tr>
<tr>
<td>500</td>
<td>Server Error</td>
<td>Something went wrong during the request was being handled. Please contact the IPC SDMS administrators.</td>
</tr>
</tbody>
</table>

When sending an invalid request, error messages will look like

```json
{
    "name": "Not Found",
    "message": "Object not found: 1234",
    "code": 0,
    "status": 404
}
```
with

- **name** the HTTP status message
- **message** a descriptive message why this error occurred
- **code** an internal code (usually 0)
- **status** the HTTP status code

Validation errors (code 422) look different and can be used to inform the client what is wrong and what needs to be done in order to perform a valid data submission. A defective submission to create a new athlete could return a list of errors like this

```json
[
  {
    "field": "npc",
    "message": "'npc' must be set."
  },
  {
    "field": "family-name",
    "message": "'family-name' must be set but can be null if athlete has no family name."
  },
  {
    "field": "date-of-birth",
    "message": "'date-of-birth' must be in format yyyy-mm-dd."
  }
  ...
]
```

Each validation error in this JSON encoded array consists of two attributes:

- **field** the attribute that failed validation
- **message** a descriptive message why validation for this attribute failed

In general, the same validation rules apply as they are used when users register new data in SDMS through its web interface to ensure integrity and consistency of the data.
6 Contact

International Paralympic Committee
Adenauerallee 212-214
53113 Bonn
Germany
Phone: +49-228-2097-200
Fax: +49-228-2097-209
E-mail: info@paralympic.org

IPC Sports Information Technology
If you have any questions, comments or enquiries for support in regard to the API v2 of the IPC Sport Data Management System, do not hesitate to contact the developer of SDMS and the SDMS API at

sdmsadmin@paralympic.org