REST API – Getting started

1. API basics

The API sets on endpoint **/[webcon portal address]/api/data** (e.g. http://dev20/WEBCONBPS/api/data) are publicly available.

By executing the GET query at **/[webcon portal address]/api/data**, you can get information about the available API versions, along with the endpoint at which they are available, and the login address. To call the API in specific version add version suffix to endpoint, e.g. **/[webcon portal address]/api/data/v1.0**.

In most API methods, the next parameter in routing is the database ID, and it specifies which content database should be used to get the resource. Here's example URL that gets user tasks from database id 1: [webcon portal address]/api/data/v1.0/db/1/tasks.

Database identifiers are available in the Designer Studio in the system settings module in content and attachments databases, available in the global parameters node.

Keep in mind that all public methods are available on **/api/data** endpoint, and updates on this endpoint are additive and do not break existing app scenarios. All other, non-public, API methods require different authentication and should not be used.

2. Authentication

To call WEBCON REST API, your application must acquire an access token, which contains information about your application and the permissions it has. To get an access token, your application must be registered in WEBCON admin panel.

Access tokens are JSON Web Tokens (JWT). They contain claims that web API use to validate the caller and to ensure that the caller has the proper permissions to perform the operation they're requesting.

To call WEBCON API, you attach the access token as a Bearer token to the authorization header in an HTTP request.

2.1. Application registration

You can manage your applications from the admin panel, which is available at [WEBCON Portal address]/adminpanel, e.g. dev20.webcon.pl/WEBCONBPS/adminpanel. To use the panel global administrator rights are required.

To register your application, follow these steps:

1. Open [WEBCON Portal address]/adminpanel in your browser and enter login details if necessary.

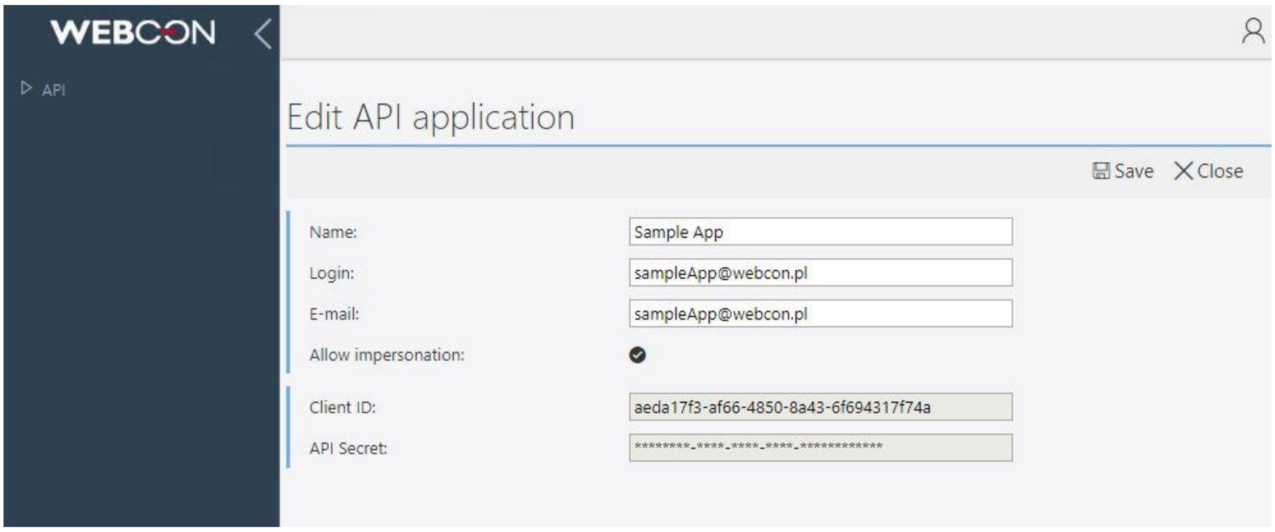
2. Choose **New API application**

3. Enter a name for the application, login in UPN format, email and check allow impersonation if needed

4. Choose **Save**

5. Save the Client ID and API Secret, they will be required to authenticate. The secret will not be shown again. If you lose it, you will have to generate a new one and the old one will stop working.

The following screenshot shows an example web application registration.



Edit API application

Name:	Sample App
Login:	sampleApp@webcon.pl
E-mail:	sampleApp@webcon.pl
Allow impersonation:	✓
Client ID:	aeda17f3-af66-4850-8a43-6f694317f74a
API Secret:	********-****-****-****-************

2.2. Get access token

To get an access token, your application has to exchange http requests with endpoint /api/login. In the body of the request you must provide the client id and client secret.

Here's an example request body:

```
{
"clientId": " aeda17f3-af66-4850-8a43-6f694317f74a",
"clientSecret": " 3640ee20-afdc-4ec0-b5af-e4b57de7aa47",
}
```

And response (the access token has been truncated for readability):

```
{
  "token": "eyJhbGciOiJIUzI1NiIsI … cD7FSf35S3--z1U7E-qrDMNu0OKi51DY32LA7s8qMlQ"
}
```

To call WEBCON API, you attach the access token as Bearer token to the Authorization header in HTTP request. Here's an example call that returns instance data with id 1:

```
HTTP/1.1
Host: dev20.webcon.pl
Authorization: Bearer eyJhbGciOiJIUzI1NiIsI … cD7FSf35S3--z1U7E-qrDMNu0OKi51DY32LA7s
GET http://dev20.webcon.pl/WEBCONBPS/api/data/v1.0/db/1/elements/1
```

2.3. Permissions and impersonation

The application uses its own permissions that you can grant in Designer Studio, just like users or groups.

Another option is to use other account permissions via impersonation, but to do so, first you need to allow the application to impersonate. This can be done in the admin panel.

To use other account you need to modify your get token request, here's an example request body with impersonation:

```
{
"clientId": " aeda17f3-af66-4850-8a43-6f694317f74a",
"clientSecret": " 3640ee20-afdc-4ec0-b5af-e4b57de7aa47",
"impersonation": {
                "login": "t.green@webcon.pl"
      }
}
```

Impersonated application uses permissions as if it were a given user.

3. Endpoints

3.1. Tasks

- **[GET] /api/data/v1.0/db/{dbid}/tasks** – returns all user's tasks from specified content database
- **[GET] /api/data/v1.0/tasks** – returns all user's tasks from all content databases

3.2. Get element (workflow instance)

[GET] /api/data/v1.0/db/{dbid}/elements/{elemid}

Returns specified instance content. Response is divided into following sections:

- **header** – contains element basic metadata such as id, instance number, version, author, step, workflow and form type information. Here's example header section:

```
"header": {
    "id": 240,
    "instanceNumber": "Api/2019/02/00017",
    "version": 3,
    "author": {
      "bpsId": "t.green@webcon.pl",
      "name": "Tom Green"
    },
    "updatedBy": {
      "bpsId": "t.green@webcon.pl",
      "name": "Tom Green"
    },
    "creationDate": "2019-02-28T10:09:05.8400000+01:00",
    "modificationDate": "2019-02-28T10:37:41.2400000+01:00",
    "enterToCurrentStepDate": "2019-02-28T10:09:07.2930000+01:00",
    "step": {
      "id": 43,
      "guid": "adb07a18-95f3-40e6-a0f0-a54552fdc88e"
    },
    "workflow": {
      "id": 12,
      "guid": "d1fd2660-669e-4f84-9e1b-88814fef3365"
    },
    "company": {
      "id": 1,
      "guid": "E554D815-F958-463A-B4DD-E2EB29B29FF2"
    },
    "formType": {
            "id": 9,
            "guid": "f1905dde-12fc-46d5-a736-0dc61e838e5b"
        }
    }
```

- **formFields** – contains listed metadata of supported element fields. Every field has its **id**, **guid**, **type**, **value** as object, **svalue** as string representation of a value. Here's example formFIelds section:

```
"formFields": [
    {
        "id": 176,
        "guid": "993c843e-112f-44ac-8c44-e8fb1fc521fa",
        "type": "SingleLine",
        "value": "Sample value",
        "svalue": "Sample value"
    },
    {
        "id": 164,
        "guid": "f4b3024b-9feb-43a1-a6b7-00f2691a9eef",
        "type": "ChoicePicker",
        "value": [
            {
                "id": "t.green@webcon.pl",
                "name": "Tom Green"
            }
        ],
        "svalue": "t.green@webcon.pl#Tom Green"
    }
  ]
```

- **itemLists** – contains metadata of a workflow instance item lists. Similar to fields it has **id** and **guid**, additionally has a list of **rows**, and each row has its **id** and list of **fields**. Here's example itemLists section:

```
"itemLists": [
    {
        "id": 3,
        "guid": "791ef030-2901-4bfb-8212-c2aa1af8aa2f",
        "columns": [
                {
                    "id": 7,
                    "guid": "E554D815-F958-463A-B4DD-E2EB29B29FF2"
                    "type": "SingleLine",
                        },
                        {
                              "id": 8,
                              "guid": "AA80F5F8-4634-4940-9E3D-3265EFDB6EB1"
                              "type": " ChoicePicker ",
                        },

                  ],
                  "rows": [
                    {
                      "id": 25,
                      "cells": [
                        {
                          "id": 7,
                          "value": "Sample value",
                          "svalue": "Sample value"
                        },
                        {
                          "id": 8,
                          "value": [
                            {
                              "id": "t.green@webcon.pl",
                              "name": "Tom Green"
                            }
                          ],
                          "svalue": "t.green@webcon.pl#Tom Green"
                        }
                      ]
                    }
                  ]
                }
              ]
```

- **comments** – list of a workflow instance’s comments, each comment has **author**, **comment** and **date**. Here’s example comments section:

```
"comments": {
      "newComment": "",
      "existingEntries": [
      {
        "author": "Tom Green",
        "comment": "Sample comment",
        "date": "2019-02-27T08:10:18.0000000+01:00"
      }
    ]
  }
```

- **attachments** – contains list of attachments metadata without their contents. Here's example attachments section:

```
"attachments": [
    {
        "id": 24,
        "name": "Sample document.docx",
        "description": "Document's sample description",
        "group": "1#Api attachments",
        "author": {
          "bpsId": "t.green@webcon.pl",
          "name": "Tom Green"
        },
         "updatedBy": {
         "bpsId": "t.green@webcon.pl",
         "name": "Tom Green"
        },
        "creationDate": "2019-02-15T11:38:47.2500000+01:00",
        "modificationDate": "2019-02-15T11:38:47.2500000+01:00",
        "version": 0
    }
]
```

- **tasks** – list of users tasks and cctasks for selected workflow intance. Here's example tasks section:

```
"tasks": {
    "directTasks": [
      {
        "user": {
          "bpsId": "t.green@webcon.pl",
          "name": "Tom Green"
        },
        "name": "",
        "description": "",
        "flag": "None",
        "isFinished": false,
        "isNew": false
      }
    ],
    "ccTasks": [
      {
        "user": {
          "bpsId": "b.szyc@webcon.pl",
          "name": "Borys Szyc"
        },
        "name": "",
        "description": "",
        "flag": "None",
        "isFinished": false,
        "isNew": true,
        "substitutions": [
          {
            "user": {
              "bpsId": "j.brodzik@webcon.pl",
              "name": "Joanna Brodzik"
            },
            "name": "",
            "description": "(Cover for: Borys Szyc)",
            "flag": "None",
            "isFinished": false,
            "isNew": true
          }
        ]
      }
    ]
  }
```

3.3. Update workflow instance

[PATCH] /api/data/v1.0/db/{dbid}/elements/{elemid}

It allows you to edit and save the values of form fields. Below is an example call that sets the values of two picker fields in two ways:

```
{
        "formFields": [
          {
                "guid": "a7fd5dd2-4de3-43be-9dda-fbdd98c6d7e8",
                "svalue": "t.green@webcon.pl#Tom Green"
          },
          {
                "id": 176,
                "value": [
                    {
                            "id": "t.green@webcon.pl",
                            "name": "Tom Green"
                    }
                ]
        }
    ]
}
```

As the above example shows, there are two different ways to choose field to edit, by **id** or by **guid**. If you provide both, guid will be used and id will be ignored.

The same is true for values, you can pass them as string using svalue or as object using value. If both were provided, value property will be used and svalue will be ignored.

Only the fields passed in the request body will change. If you want to clear the field value, pass null to the value parameter. For example:

```
    {
            "guid": "a7fd5dd2-4de3-43be-9dda-fbdd98c6d7e8",
            "value": null
    }
```

3.4. Move workflow instance to next step

[PATCH] /api/data/v1.0/db/{dbid}/elements/{elemid}

Allows you to edit and move instances to the next step. Path can be specified by three different flags:

- **?pathId={id}** – path is selected by id
- **?pathguid={guid}** – path is selected by guid
- **?defaultpath=1** – moves the instance with the default path

Body parameters for editing fields are the same as for the update method. In case if several parameters are passed simultaneously, the guid is the first one to be taken into account, the next is the id. Default is used only if no other parameter has been passed.

3.5. Start new workflow instance

[POST] /api/data/v1.0/db/{dbid}/elements

Allows you to start a new instance. The path can be specified in the same way as in the move method, using flags **?pathId={pathid}** for id, **?pathguid={guid}** for guid and **?defaultpath=1** for default path. If more than one parameter is specified, the guid is the first one to be taken into account, the next is the id.

Below is an example call that starts an instance using workflow with the id 1, and a form type with the id 1:

```
{
        “workflow”: {
                “id”: 1
        },
        “formType”:{
                “id”:1
        },
        “formFields”: [
                {
                        "guid": "a1f7dd0d-e91f-4ba4-aa4a-1dfc884db243",
                        "svalue": "Sample text"
                }
        ]
}
```

Fields that are not specified but are visible will remain empty or will be given default values from the form field’s configuration.

3.6. Attachments

- **Get attachment**

  [GET] /api/data/v1.0/db/{dbid}/elements/{elemid}/attachments/{attid} – returns specified attachment metadata and content as base64 string

- **Get attachment’s content**

  [GET] /api/data/v1.0/db/{dbid}/elements/{elemid}/attachments/{attid}/content – returns selected attachment content as base64 string

- **Add new attachment**
  [POST] /api/data/v1.0/db/{dbid}/elements/{elemid}/attachments – allows you to add attachment to an existing document. Below is example call (the file content has been truncated for readability):

  ```
  {
      "metadata": {
          "name": "SampleFile.txt",
          "description": "Sample file description",
          "group": "1#Api attachments"
      },
      "content": "77u/ … AgIH0NCiAgfQ0KfQ0K"
  }
  ```

- **Update existing attachment**
  [PATCH] /api/data/v1.0/db/{dbid}/elements/{elemid}/attachments/{attid} – allows

you to update existing attachment. The method accepts the same parameters as add new attachment method, any combination of parameters can be passed.

4. HTTP methods

WEBCON API uses the HTTP method on your request to determinate what action should be performed. The API supports following methods:

- GET – read data from a resource
- POST – create new resource or perform an action
- PATCH – update a resource with new values

For the method GET no request body is required.

The POST and PATCH methods require a request body specified in JSON format, that contains additional information.

5. Fields and item lists

5.1. Guid vs id

There are two ways of specifying field in API, by **ID** and by **GUID**. GUID is main property, so if you provide both system will use GUID.
In most cases, we recommend using GUIDs because they are unchanged, you can be sure that the application will refer to the same fields after migrating processes between different environments.

5.2. Value vs svalue

There are also two ways of passing field values. By its string representation using **svalue** property, and by its object representation using **value** property. Value is the preferred property, which means it will be used first if both are provided.

The field value can be cleared by passing null as the value.

5.3. Form fields examples

List of example value and svalue for all fields:

- **Single line of text:**

```
"value": "Sample value",
"svalue": "Sample value"
```

- **Single line of text – hyperlink field:**

```
"value": {
        "url": "www.webcon.pl",
        "name": "Webcon"
},
"svalue": "link:www.webcon.pl;displayname:Webcon"
```

- **Single line of text – image field:**

```
"value": {
        "url": "www.webcon.pl/Logo.jpg",
        "name": "Logo"
},
"svalue": "image:www.webcon.pl;name:Webcon"
```

- **Email**

```
"value": "t.green@webcon.pl",
"svalue": "t.green@webcon.pl"
```

- **Date and time**

"value": "2019-02-06",

"svalue": "2019-02-06"

- **Date and time – time selection enabled**

  "value": "2019-02-27T14:00:35.4470000+01:00",

  "svalue": "2019-02-27T14:00:35.4470000+01:00"

- **Boolean**

  "value": true,

  "svalue": "1"

- **Choice tree**

```
"value": {
            "id": "t.green@webcon.pl",
            "name": "Tom Green"
},
"svalue": " t.green@webcon.pl#Tom Green "
```

- **Choice field** and **Person or group**

```
"value": {
            "id": "t.green@webcon.pl",
            "name": "Tom Green"
},
"svalue": " t.green@webcon.pl#Tom Green "
```

- **Choice field** and **Person or group – multiple values**

```
"value": [
            {
              "id": " t.green@webcon.pl ",
              "name": " Tom Green "
            },
            {
              "id": "l.tyler@webcon.pl",
              "name": "Liv Tyler"
            }
],
"svalue": " t.green@webcon.pl#Tom Green;l.tyler@webcon.pl#Liv Tyler"
```

- **Multiple lines of text**

  ```
  "value": "Sample multiline value\nsecond value line",
  "svalue": "Sample multiline value\nsecond value line"
  ```

5.4. Item lists basics

Item lists are presented in a separate property called "**itemLists**". It can be passed to every instance related method (get, update, move, start new). Each item list has its **rows** array, and each row has **fields** array. Below is an example of a call:

```
"itemLists": [
    {
        "guid": "5b78500d-d520-407c-a906-bb12b9a53f04",
        "rows": [
            {
                "cells": [
                    {
                        "id": 25,
                        "value": "Sample text"
                    },
                    {
                        "id": 29,
                        "svalue": "t.green@webcon.pl#Tom Green"
                    }
                ]
            }
        ]
    }
]
```

Passing values of individual columns works in the same way as in standard form fields. It is enough to pass only selected column values.

5.5. Operations on item lists

Operations on the list are based on the transmission of differences. If the row or column has not been submitted in the query, it will remain unchanged.

- **Add a new row**

  While row id is not specified, a new row will be added with the values passed:

```
"itemLists": [
    {
```

```
        "guid": "5b78500d-d520-407c-a906-bb12b9a53f04",
        "rows": [
          {
            "cells": [
              (field values can be passed here)
            ]
          }
        ]
      }
]
```

- **Edit an existing row**

  While row id is provided, passed values will override existing values in selected row:

```
"itemLists": [
      {
        "guid": "5b78500d-d520-407c-a906-bb12b9a53f04",
        "rows": [
          {
            "id": 265,
            "cells": [
              (field values can be passed here)
            ]
          }
        ]
      }
]
```

- **Delete an existing row**

  While row id is provided, and delete flag is set to true, row will be deleted:

```
"itemLists": [
      {
        "guid": "5b78500d-d520-407c-a906-bb12b9a53f04",
        "rows": [
          {
            "id": 265,
            "delete": true
          }
        ]
      }
]
```

All of the above operations can be combined to allow full control over the list values.

5.6. Visibility, editability and requiredness

API returns only the fields visible to the application or user which the application is impersonating.

If some of required fields have not been passed in the request or do not have a default value, response will return an error – 400 bad request.

If you try to change values of read-only fields, you will get the response - 409 conflict.