

ctrlX CORE: REST API

Insights - ctrlX AUTOMATION: Engineering

August 2020



ctrIX CORE

Data Layer

The ctrIX Data Layer forms the backbone of ctrIX CORE. The Data Layer, the control's data interface, includes data and other functionality which may be consumed or modified by a suitably qualified client.

In this article we give a brief overview of some of the data that is available in the ctrIX Data Layer and how it may be accessed. As we shall see, the underlying mechanism is such that apps, whether running onboard the control, on a smart phone, on an external PC or in the cloud, can interface with the Data Layer using the same general approach.

When an app is installed on ctrIX CORE, it may publish data (or, more generally, expose *functionality*) to the Data Layer. Such data appears as a clearly identifiable node within the Data Layer tree. For example, with the PLC and MOTION apps installed, the Data Layer looks like this:

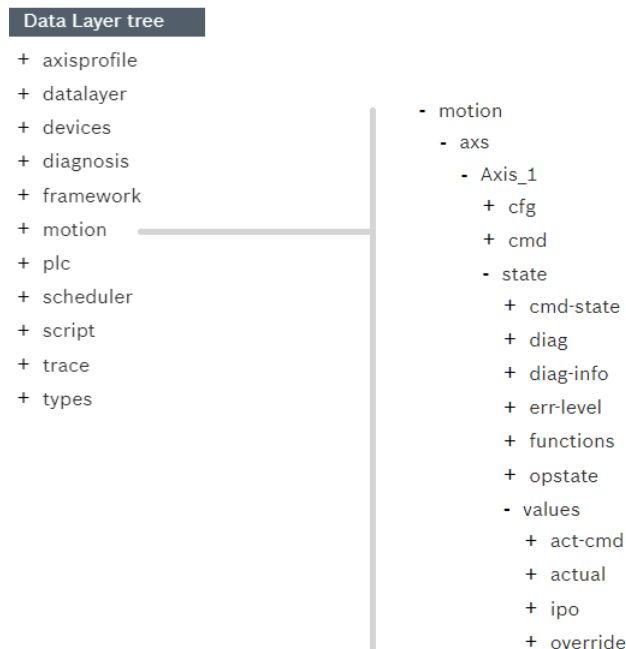


Figure 1: The Data Layer tree as it appears in the ctrIX CORE web interface. Nodes **+ motion** and **+ plc** are directly associated with the MOTION and PLC apps resp. and will only appear once these apps are installed.

All data or functionality exposed by a given app is available within the associated node. For example, under **+ motion**, we find a listing of all configured axes together with critical data, including diagnostic information, command and actual position values and operation state.

API

An application programming interface, or API, is a set of rules or methods that define how external applications may interact with the application itself. We often think of the original application as the *server* and the external applications as *clients* and for simplicity we'll adopt this language here.

The API defines what data may be accessed, whether it may be modified, etc. and how each of these operations is performed. In other words, it defines the set of requests that may be made by the client and how to make them. We often blur the API, the abstract definition of this functionality, with the library that implements the API; in most cases this doesn't create too much trouble or confusion.

An example of an API, now well known to the Rexroth community, is the Open Core Interface. Through this interface, software clients built on a various platforms - Windows, LabVIEW, Matlab, iOS, Android, etc. - can access data or other functionality on IndraMotion controls or drives.

Ebay API

Ebay, the well-known auctioning website, introduced its API in 2000. Prior to this point, all Ebay transactions occurred directly on Ebay's website. The API allowed developers to build custom, narrowly-focused auctions on third-party websites or applications, providing a better user experience. At the time, the Wall Street Journal wrote¹:

Having captured most of the Internet auction market with its own site, eBay Inc. has a new goal: to become an "operating system" for e-commerce on the Web.

For the past six months, the company has been developing technology that will let Web companies display eBay auctions – whether listings for baseball cards, stamps or automobiles – on their own independent sites, which could greatly expand the visibility of eBay auctions among potential buyers.

Ebay was an early adopter of a special kind of API, now commonplace on the Web, called REST. A REST API, meaning a programming interface that follows the REST design pattern, exposes a set of *resources* via a fixed set of *resource methods*. The REST design pattern also constrains the interface in ways that promote uniformity and clarify the roles of client and server. The details are beyond our scope here, but suffice it to say, the ubiquitous adoption of the REST design pattern has imposed on Web services a common workflow or *look and feel*.

A typical REST API roughly follows the pattern shown in Figure 2. Resources are represented by Uniform Resource Identifiers (URI) within the server of the form `https://host/path/query` and the resource methods are request types associated with the HTTP protocol: GET, PUSH, PUT, DELETE, etc. Note that the client always makes requests for data access or modifications and that the server may deny these requests based on various conditions, including lack of credentials or current server state.

¹<https://www.wsj.com/articles/SB974675427606513763>

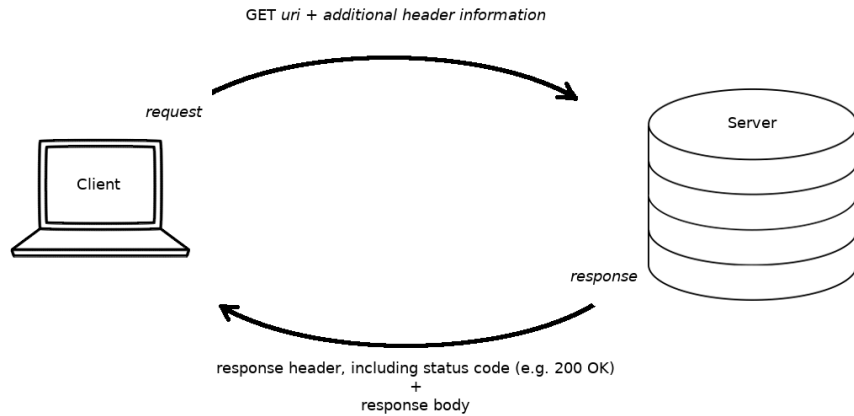


Figure 2: Typical REST API workflow: Client issues request for data associated to a defined resource (URI). If properly authenticated, server responds with requested data. Note that other request types (e.g. PUSH, PUT) may require additional data in a request body.

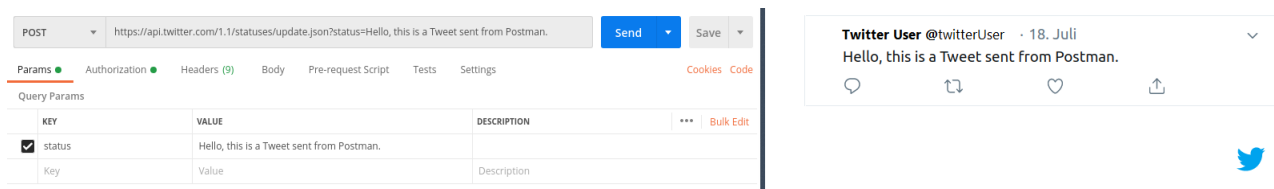
Like Ebay, Twitter also has an API, part of which is RESTful. For example, using Twitter’s REST API, third parties can search the Twitter database by user or by hash.

To give a concrete example, making a GET request to <https://api.twitter.com/1.1/search/tweets.json?q=from%3Aboschrexrothus>, allows us to access the most recent tweets sent by @BoschRexrothUS.

The image shows two side-by-side screenshots. On the left is a screenshot of the Postman REST client interface. The top bar shows a GET request to `https://api.twitter.com/1.1/search/tweets.json?q=from%3Aboschrexrothus`. The response status is 200 OK. The response body is shown in JSON format, with a red box highlighting the 'text' field of the first tweet: `"text": "Lean manufacturing is a snap with the world's largest modular profile system from Bosch #Rexroth. Quickly assemble. https://t.co/n99gdKttAx"`. On the right is a screenshot of a tweet from Bosch Rexroth Corp (@BoschRexrothUS) dated 17. Juli. The tweet text is: `Lean manufacturing is a snap with the world's largest modular profile system from Bosch #Rexroth. Quickly assemble workstations, flow racks and material shuttles — virtually anything you need to boost productivity. Get details. bit.ly/3fuBToz`. Below the text is a photo of a modern office interior with large windows and a reception desk. The Bosch Rexroth logo is visible in the bottom right corner of the tweet image.

Figure 3: Using the Twitter API, clients may access the recent activity of a specific user. Note that additional authentication information must be supplied in the request header. The REST client shown on the left is Postman.

Similarly, we can post to Twitter (i.e. "tweet") using a POST request:



The image shows a Postman interface on the left and a simulated Twitter tweet on the right. The Postman interface displays a POST request to the URL `https://api.twitter.com/1.1/statuses/update.json?status=Hello, this is a Tweet sent from Postman.` with a query parameter `status=Hello, this is a Tweet sent from Postman.` The tweet on the right is from `Twitter User @twitterUser` and contains the text `Hello, this is a Tweet sent from Postman.`

Figure 4: The Twitter API also allows clients to tweet to associated user accounts using a POST request. (@twitterUser is not a real user.)

It is worth noting that, in both cases, successful requests to the Twitter API require proper authorization². Access to specific resources is given only to qualified users.

²For an overview of the Authorization requirements, see Twitter's development site: <https://developer.twitter.com/en/docs/basics/authentication>

Examples

In a similar vein, ctrlX CORE includes a REST API, allowing qualified users to access the Data Layer. To give a flavor for the type of information or functionality that can be accessed, we list some simple examples. Throughout, 192.168.1.1 is the IP address of the ctrlX CORE control.

1. To get a listing of the main items in the Data Layer, issue a *GET* request to `/automation/api/v1.0/?type=browse`. Compare the array labeled "value" below to the tree shown in Figure 1.

Request

```
GET
https://192.168.1.1/automation/api/v1.0/?type=browse
```

Response (body)

```
{
  "type": "arstring",
  "value": [
    "axisprofile",
    "datalayer",
    "devices",
    "diagnosis",
    "framework",
    "motion",
    "plc",
    "scheduler",
    "script",
    "trace",
    "types"
  ]
}
```

2. For general system information, issue a *GET* request to `/systeminfo/api/v1.0/systeminfo`.

Request

```
GET
https://192.168.1.1/systeminfo/api/v1.0/systeminfo
```

Response (body)

```
{
  "Hostname": "VirtualControl-1", "IpAddress": "192.168.1.1",
  "OperatingSystem": "Ubuntu Core 18", "Architecture": "amd64",
  "MACAddress": "de:ad:be:00:00:01"
}
```

3. To read the current values of an axis configured on the control (here Axis_1), issue a *GET* request to `/automation/api/v1.0/motion/axs/Axis_1/state/values/actual`.

Request

```
GET
https://192.168.1.1/automation/api/v1.0/motion/axs/Axis_1/state/values/actual
```

Response (body)

```
{
  "actualPos": 452.8149999999993,
  "actualVel": 0.0,
  "actualAcc": 0.0,
  "actualTorque": 0.0,
  "distLeft": 0.0
}
```

4. The REST API also allows us to change the control's state. For example, we may reset the axis above by issuing a *POST* request to `/automation/api/v1.0/motion/axs/Axis_1/cmd/reset`. (The response body is empty in this case.)

Request

```
POST
https://192.168.1.1/automation/api/v1.0/motion/axs/Axis_1/cmd/reset
```

Response (body)

```
empty
```

5. Assuming the control is not in its Running state, we can also add or delete Axes. To add Axis_3, for example, issue a *POST* request to `/automation/api/v1.0/motion/axs` with the request body shown below.

Request

```
POST
https://192.168.1.1/automation/api/v1.0/motion/axs
```

Request (body)

```
{"type": "string", "value": "Axis_3"}
```

6. To delete this axis, issue a *DELETE* request to `/automation/api/v1.0/motion/axs/Axis.3`.

Request

```
DELETE
https://192.168.1.1/automation/api/v1.0/motion/axs/Axis_3
```

Response (body)

```
empty
```

7. Finally, to read a PLC variable, `dTest`, declared in program `PLC_PRG` (and included in the symbol file), issue a *GET* request to `/api/v1.0/plc/app/Application/sym/PLC_PRG/dTest`.

Request

```
GET
https://192.168.1.1/automation/api/v1.0/plc/app/Application/sym/PLC_PRG/dTest
```

Response (body)

```
{
  "type": "int32",
  "value": 332
}
```

8. To write to this same variable, issue a *PUT* request to the same URI with a suitably configured request body.

Request

```
PUT
https://192.168.1.1/automation/api/v1.0/plc/app/Application/sym/PLC_PRG/dTest
```

Request (body)

```
{
  "type": "int32",
  "value": 333
}
```


As in the case of the Twitter, successful requests to the ctrlX REST API require strict authorization. All of the requests described previously require an authorization token. To obtain one, submit a *POST* request to the internal identity manager with configured username and password as shown below.

Request

```
POST
https://192.168.1.1/identity-manager/api/v1.0/auth/token
```

Request (body)

```
{
  "name": "username",
  "password": "*****"
}
```

Upon success the requester is issued a bearer token which should be added to the header of all subsequent API requests.

Reply (body)

```
{
  "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE1OTU5NjgwODcsIm1hdCI6MTU5NTkzOTI4NywiawQiOiIxMDAxIiwibmFtZSI6ImJvc2NocmV4cm90aCI6Im5vbmNlIjoimTI2MTdlMzgtODI5OS00ZmFkLTk0M2QtOWU5YjY1ZGI5YjYyIiwicGxjaGFuZGx1IjoxLCJzY29wZSI6WyJyZXhyb3RoLWRldmljZS5hbGwucnd4Ii119.GmHc4RIGMDfkY2d4Ws2X4zWmBanq1Uq77mhcX1zdhWvfNsYP0Q3g_DS1rc3U8KGyz4DCpl1W3iIzOpLsfNZykPwJ5MEJv45nEmAYms8SVPJAdIXVFfr_vB4jKehdxocANwuonIWGX_PpOWcwDs7HyZJZQy5nMN1EXj8z-5e5UOd2Peo6q_vZLf6OnoM7V8FmkWFNmeSrQ6mEFvhJkcUDybgsFpLIyOhAR0kvfUu3d94r3UUJSp3L8k86a36tQcWsP8mB1_E0XMMxNYEss4ERtsWab8j4rihEpfhAujaX8ELeNgLdlPdy4eoiz_ikU6Uoa24Cb1zL8t0tChOQPtu33VduiLqavG1Rv9qnPm oZGj5l jy6CoZGQtwTwU1CCVRUuTzWFLcOhHNL EegKrMQeDmc8wscWsGwEewGHiepdWQzVJ7pLS813VXl2ULscqTD_h4fLRhOawHbcptmREpkyMCUc3S5FNmXEjEfecp7KmVCvsOUI_ITP9ocW1eCa49KRmnrbtfgp5soiSLy7yfrD5dOhjEPp9Gw42jWMZTF9WkFQGV8u09GFUIZ24mCv17BV5wPsQZGaxHPFmGxldcGQRSEHsiDGv0OxAvqWOJ7WhmEpwLfhwkRkVett40CfiBJxuGH0mSpjtCyOnQP2pfp1R1zOu9QsW8-i0-xQ6Y",
  "token_type": "Bearer"
}
```

Finally we note that ctrlX CORE also allows administrators to restrict user access to the Data Layer in a granular way, meaning that a particular user may be granted access to only a limited part of the API.

Why REST?

Virtually all modern programming languages support functionality that allow users to access a Web server via its REST API. The table below lists some of the most popular programming languages³, all of which support, either natively or through third-party resources, the required functionality.

<i>Language</i>	<i>HTTP client support</i>	<i>Library, module or resource</i>
JavaScript	Y	Fetch, Axios, ...
Python	Y	Lib/http.client.py
Java	Y	OpenJDK: HttpClient
PHP	Y	Guzzle, Httpful, Requests, ...
C#	Y	System.Net.Http: HttpClient
C/C++	Y	libcurl, Beast (Boost), ...
TypeScript	Y	Fetch, Axios, ...
Ruby	Y	net/http (standard), Faraday, Curb, ...

The ctrIX CORE REST API thus gives programmers broad freedom in choice of language when configuring the non-realtime functionality of the control or consuming data published by the Data Layer. By building the ctrIX CORE's programming interface on standardized web technologies with near universal support, Rexroth has created an automation platform well positioned to play an integral role in the Factory of the Future.

Michael Schaefer (DC-AE) provides a short introduction to the ctrIX CORE REST API, including configuration of the request header, in a tutorial video found here:

<https://e.video-cdn.net/video?video-id=EKqFyBxSHZKiPfJti4W1w9&player-id=B2xD9x5D-UTrfT84ZR61rZ>

We will preview the ctrIX CORE realtime API in a future issue of the ctrIX eNewsletter.

Carl Bostrom
Application Engineering (DCNA/SAE22-US)

³See, for example, <https://octoverse.github.com/>