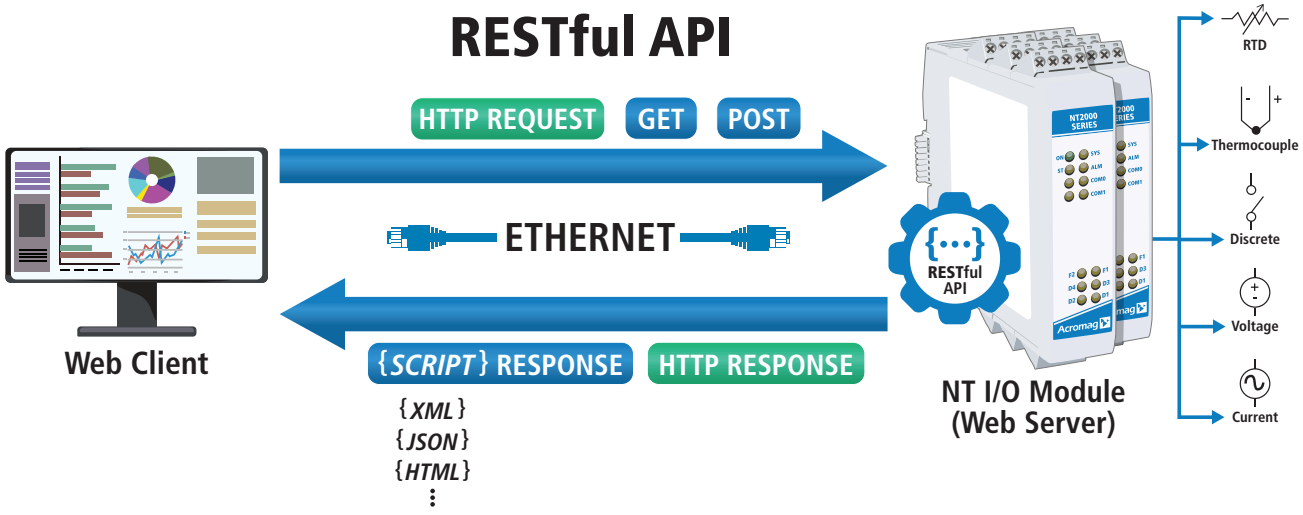
Application Note

Acromag
THE LEADER IN INDUSTRIAL I/O

Introduction to the BusWorks NT Series I/O System
RESTful Application Program Interface (API)



Bulletin # 8501-226A

Tel 248-295-0880 ■ Fax 248-624-9234 ■ sales@acromag.com ■ www.acromag.com ■ 30765 Wixom Rd, Wixom, MI 48393 USA

Contents

Introduction to the BusWorks NT Series I/O System RESTful Application Program Interface (API)

The Acromag NT I/O system is an expandable modular I/O system that allows users to mix and match up to four different types of I/O interface groups (18 varieties) to measure, convert, amplify and isolate I/O signals. This allows each NT *system* to be tailored to a user's own unique industrial application with network support for both wired and wireless Ethernet interfaces.

Eighteen NT I/O modules address variations and combinations of digital/discrete I/O, current/voltage I/O, Thermocouple/millivoltage input, RTD/resistance input, and AC voltage/current input.

Table 1: Eighteen Compatible NT I/O Model Groups for NTE/NTW and NTX Modules

NTE or NTW CPU and I/O Main Module	NTX I/O Expansion Module	SUPPORTED I/O CHANNELGROUP/MODULE (NT systems support 1-4 I/O Groups)
NT_2111	NTX2111	16CH DI/O Sinking
NT_2121	NTX2121	16CH DI/O Sourcing
NT_2711	NTX2711	Stepper Motor Controller DI/O
NTE2131	NTX2131	6CH Mechanical Relay & 6CH DI Active High
NT_2141	NTX2141	6CH 130V AC Input/2CH DI/O Sinking
NT_2142	NTX2142	6CH 240V AC Input/2CH DI/O Sinking
NT_2211	NTX2211	8CH Differential Current & 2 DI/O Sinking
NT_2221	NTX2221	16CH Single-Ended Current Input
NT_2231	NTX2231	8CH Differential Voltage Input & 2 DI/O Sinking
NT_2241	NTX2241	16CH Single-Ended Voltage Input
NT_2311	NTX2311	8CH 0-22mA Current Output
NT_2321	NTX2321	8CH Voltage Output w/Common Return
NT_2511	NTX2511	4CH Diff I+4CH DI/O SRC+2CH AO-I Combo
NT_2531	NTX2351	4CH Diff V+4CH DI/O SRC+2CH AO-I Combo
NT_2512	NTX2512	4CH Diff I+4CH DI/O SRC Combo (no AO-I)
NT_2532	NTX2532	4CH Diff V+4CH DI/O SRC Combo (no AO-I)
NT_2611	NTX2611	8CH Thermocouple/mV & 2 DI/O Sinking
NT_2621	NTX2621	4CH 2/3/4-Wire RTD/Resistance & 2 DI/O Sinking

The first NTE/NTW system module is represented as I/O slot 0 of 4 available I/O slots and its I/O board is mated with the system network/CPU inside the main module housing (an NTE or NTW module). You may optionally connect 1-3 more NTX expansion I/O modules in any mix by plugging them together with the main NTE/NTW module moving left to right along its DIN rail communication bus (I/O slots 1-3 in any mix). Each system slot 0-3 represents a unique channel I/O group per model definition above to potentially support from 6 to 48 I/O channels at a single system IP address.

The NTE2xxx supports two wired Ethernet ports, while the NTW2xxx supports one Wireless Ethernet port and one wired Ethernet port (xxxx represents the 4-digit number that identifies its embedded I/O mix as shown above). For program access, the NT has an integrated **A**pplication **P**rogram **I**nterface (**API**) that allows you to issue network command lines to read/write its I/O data using HTTP protocol commands over Ethernet.

What is a RESTful API?

A **REST**ful API refers to **RE**presentational **S**tate **T**ransfer *Application Program Interface*. It's a method of client-server communication widely used by web-based services, web-applications, and other web-based systems like the NT to allow them to easily and securely communicate with each other over the world-wide web along their common Ethernet interface. RESTful denotes an API that follows the popular REST architectural style which delineates server resources by modeling **URI**'s or objects for each resource a server wishes to expose to the network, along with its required data format (the data object is modeled by the server to a network client). Web servers that implement REST architecture are considered RESTful web services and will utilize RESTful API's which only use HTTP requests to interact with object data. A REST API and a RESTful API are both used to build web applications, but the traditional RESTful API of the NT has the added distinction of allowing the NT server to be accessed through the HTTP protocol of Ethernet, not requiring any additional server logic.

The RESTful API itself is not really a protocol or standard, but does conform to specific REST architectural constraints, like the use of a *uniform interface* (Ethernet) and *stateless communication* (allowing the server to not maintain any other state that might result from any previous interaction with a client), the use of using *cacheable data* (its ability to store copies of frequently accessed data so that when a client makes a request, it first goes through its *cache*, then on to the server), and its use of a *client-server request/response model* (cache is simply a system of storing and retrieving network requests and their corresponding responses). These restraints coupled with self-descriptive messages and even hypermedia make the RESTful API very attractive and easy to use in web development.

Now you may be wondering how you may utilize the NT RESTful API. Note that it is not intended for use with cell phones, tablets, or the cloud, but you can create your own device application or write scripts to send the necessary HTTP Get/Post commands to accomplish that as illustrated in the examples that follow, which do not advocate for any specific *application*, but simply introduce the NT RESTful API that you may use in your own *application*, perhaps by simply sending script messages or within an existing application like Postman (Postman is a software development platform that helps you collaborate with others to design, build, test, and use API's). Of course, the NT implements a RESTful API, so there needs to be a connection between a Client and the NT I/O system (server) along Ethernet, either a wired connection (NTE system), or wireless connection (NTW system).

Note that a "script" refers to an instruction or sequence of instructions, often written in a programming language, typically to perform a specific task within a larger software program. Scripts are usually executed on-the-fly and utilize an interpreter to perform them rather than being compiled beforehand as a software program is. Think of scripts as small programs that can be easily run within another application or program environment, or using a scripting language like JavaScript, Python, or PHP. Compared to software coding which involves creating a software program or software application as a sequence of program instructions that are compiled and converted to machine code to be carried out by a computer processor. Scripting is simpler and involves automating tasks or processes using *scripts* which are a program or sequence of instructions interpreted or carried out by some other means or another program rather than by compiling them to machine code to be executed by a computer processor. Application programs are typically more complex and feature-rich, requiring extensive development efforts and the resources of integrated processors. The script is simpler and *lightweight*, usually written to accomplish a single task very efficiently and *on-the-fly*.

There are several *script* languages targeted to web applications that can interpret script requests like PHP, Python, JavaScript, and jQuery which are widely used by millions of internet users every day. JSON (JavaScript Object Notation) is one such script language that has a text-based format for storing and exchanging data and is commonly used in web development, data analysis, and software engineering. JSON is often used on the web when data is sent from a server to a web page. The RESTful API requests supported by the NT may optionally be invoked using JSON.

Of course, it's not necessary to write a program to invoke an NT RESTful API script request, you can more easily use a command line tool (much like a terminal emulator) to send the HTTP request over Ethernet to the NT server and receive its response, one line at a time. The NT RESTful API implementation is very simple and uses only the HTTP methods of GET and POST for accessing data (it does not use PUT instead of POST because PUT is intended for creating a new data resource and this is not required by the NT).

As suggested earlier, you could choose to write your own mobile application program that allows you to send HTTP commands/requests in automated fashion from a cell phone or tablet, but this is beyond the scope of this APP note which just provide example NT scripts that you can get started with using a simple desktop/laptop as a *client* device to exchange NT I/O information over Ethernet with an NT system device (*server*).

One common command line interface inherent to many *client* Operating Systems including the Windows OS is cURL, which denotes Client URL command line tool which can send single-line scripts to the NT and return the data requested. cURL is just one of a myriad of available tools that can send Ethernet HTTP requests and illustrated in the examples provided later in this paper. Of course, you can always choose to embed Request scripts as modeled in some other larger software program, or combine them as a sequence of program instructions, and those will ultimately be carried out under control of a native processor within a development environment that also includes an interpreter to discern scripts from program instructions embedded within a software program.

Introduction to The NT RESTful Application Program Interface (API)

The NT I/O system itself is a combination of field inputs that can be read, and/or outputs that can be adjusted or turned ON/OFF—together these are the NT server's I/O resources. The NT I/O system firmware embeds a RESTful **API** (**A**pplication **P**rogram **I**nterface) to support reading/writing these NT I/O resources by modeling them as server objects that may be accessed over Ethernet using a set of rules set forth by its RESTful API that any network ***Client*** must follow to communicate with the NT I/O ***Server*** *programmatically*. A *Client* is any user or software application on a network that wishes to access exposed Server resources and for the NT, this is along its Ethernet network interface, wired or wirelessly.

The NT RESTful API acts as a web service that is called by a network client which drives the NT server to return a *representation* of a particular resource's *current* state back to the client system. That is, the RESTful API client initiates a change in its own state through a server request sent to the server using a representation model of its current state to transfer the data back and forth that is not specifically tied to the resource itself or to the HTTP method used (this allows an API to handle many different types of requests and return data in multiple formats). For the NT, software developers may access a subset of

server resources using HTTP methods and the structure of the response received will be based on the referenced resource object modeled by the API's URI. The RESTful API of the NT URI employs an object model that identifies the specific resource and its required data to allow any client to access it by sending an HTTP request along with a representation of the object's resource and its own data in the same format as the actual server response but allowing the server to return the *current* data in its request response.

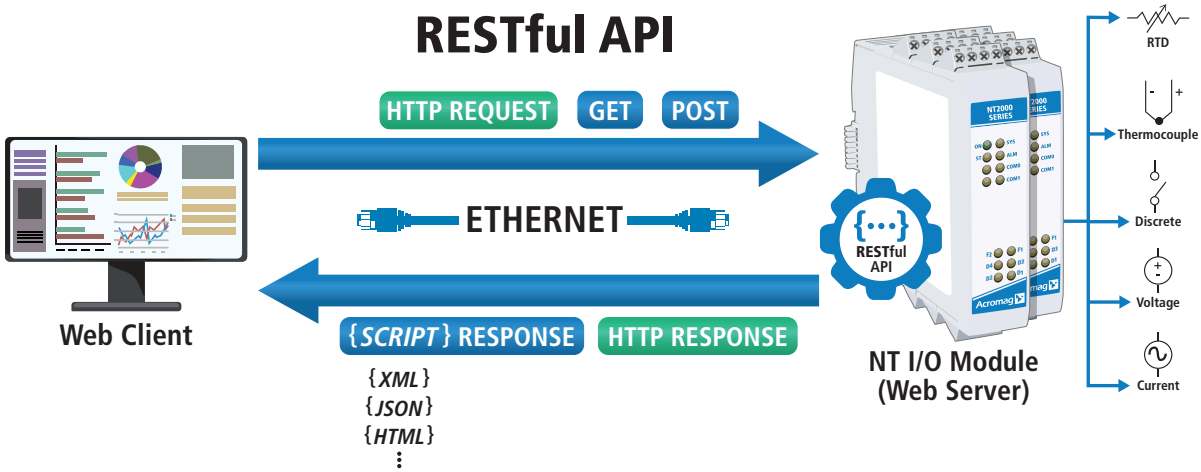
The NT RESTful API allows other web applications on the internet to use the Ethernet's inherent HTTP web-protocol (Hypertext Transfer Protocol) which is another set of rules that web-browsers, web-clients, and web servers use to simply communicate with each other (between servers and clients) to exchange information or simply load web pages. The RESTful APIs of connected clients and servers each invoke HTTP protocol requests to gain access, and retrieve and modify application data or gather system information over their common Ethernet network interface. That is, an NT user/client would initiate an API command/call to an Acromag NT system to request it do something (or to access its resource), with the NT API operating as the *middleman* between the client and NT I/O server. The NT API will respond to the client's **HTTP** request while utilizing only HTTP methods of **GET** (Retrieve data from the server) or **POST** (Send data to the server) to read and write NT server I/O data, similarly as browsing information over the internet. The NT server returns a response including the I/O data requested. Of course, RESTful API data exchanges using HTTP are not limited to software-software exchanges as illustrated, but also useful for machine-to-machine and software-machine exchanges too.

The RESTful API is the perhaps the most popular web interface method because it relies solely on the shared universal HTTP protocol standard of Ethernet making it format-agnostic in the sense that it can be used *unchanged* across many different web development environments without having to modify itself—you may use XML, JSON, HTML, or another language to build it and it doesn't change form, very useful for mobile app projects, IoT Internet of Things devices and more. The use of RESTful API's are very efficient, plenty fast, and will consume less network bandwidth operating "on-the-fly" and independent of native processors. This is very cost-effective in the sense that varied developers can use the same API request typed exactly as illustrated across many environments without having to employ third-party tools or specific processors.

An API Command Line of a RESTful web service will denote both its URL **Uniform Resource Locator** (web address of the server/port) and the specific command via a URI (Universal Resource Identifier) to front the resource request and its response.

You are already familiar with the use of web addresses or URL's when you visit a website to access static content like a web page, a program of a web page, an image or video and most web browsers display the URL of the web page visited using a typical form *http://www.example.com/index.html* above the returned content. That is "http:" indicates the hyper-text transfer protocol, the host name www.example.com, and the file name index.html. URL's may only be denoted using characters from the standard 128-character ASCII set and many special characters like spaces and slashes are not permissible. Some other RESTful API web service requests may even include a URL in their response to *hyperlink* another RESTful resource or web location.

The **URI** of a RESTful API is another unique sequence of characters that is used to distinguish one resource from another at the specified URL address and may include special characters like ~ or !. In that sense, it can use the HTTP protocol to point to *any type* of resource, including another web resource or web location, allowing one resource to interact between and among other resources within the device API itself or an API hosted on a different server without knowing that server's specific URL address. It may even point to another resource with a different required *scheme* than http such as **https** which is alternately accessed an appended "s" to denote the use of Transport Layer Security (TLS). Good RESTful URI design should not use query parameters to alter a resource's state and must remain stateless and should avoid mixed-case path definitions (the use of lowercase is preferred), or implementation-specific extensions within the URI. URIs are also persistent and do not change over time like URLs. Both URL's and URI's follow the same RFC 3986 specification. The RESTful API URL locates a resource by its unique IP address and its URI will identify a unique resource at that address or refer a client to another API web resource using a hypermedia link. Of course, NT RESTful API URI models are much simpler and will only access NT system resources without hyperlinking. The following diagram illustrates a possible exchange of I/O data between a network client and NT server over Ethernet.



Introduction to the two HTTP Request Methods Used by the Simplified NT RESTful API

The NT series RESTful API provides an interface for users (clients) to communicate with remote NT I/O systems (servers) using HTTP command requests to allow it to monitor and control NT system I/O data (tasks typically gather important data from connected I/O slots or modify output channel data). REST API design format uses a stateless request model with HTTP requests that work independently in a single line and may be invoked in any order such that keeping transient state information between requests is not feasible.

The NT uses a simplified RESTful API which only utilizes two common REST methods of the HTTP protocol to read and write the server's resource data: **POST and GET**. *GET* is used to retrieve resource data from the server at the specified URL address (GET is read-only). The *POST* method is used to send data to the server to modify or update a data resource at the server or to trigger server-side processing routines that don't update anything. The POST URI identifies the object on the server that can process included data. You must invoke the NT RESTful API Request exactly as modeled—do not substitute or omit any characters or change any character's case.

In the NT examples below, keeping things simple we will use "curl" or cURL denoting use of the **client URL**, a common open-source client command line interface tool (CLI) that you can use to make simple network requests on a line-by-line basis to transfer NT server data along an Ethernet network connection. Basically, cURL lets you talk to a *server* by specifying its location in the form of a URL address, and the data you want to send/receive in the form of a URI model defined by the NT RESTful API. There are many CLI (Command Line Interface) tools that you could use besides cURL and these are also text-driven and capable of sending the modeled URI command lines to GET NT resources using the HTTP methods described and the request model format typed exactly as shown, unchanged between tools.

Usually a curl Command Line Interface (CLI) comes pre-installed in Windows with the necessary environment to run cURL commands. You can Simply type the ``Win + R'' key combination, then type 'cmd' to open the command line tool and then input "curl –help" to check for the cURL installation status of your Windows OS installation.

How you would execute a curl script to test your Windows curl installation:

1. Launch your particular CLI command-line interface.

In Windows, open the **START** menu, type **cmd** in the sarch box and press **Enter**. ...

2. Build your cURL statement.

Build your cURL statment using a text editor, then copy this curl statement from your text file and paste it at the cmd command prompt.

3. Press Enter to run the curl statement.

Note: The cURL command specifies the server URL address (not client URL) and optionally the data you want to send, through the Command Line Interface (CLI) cURL.

***TIP:** You can usually invoke **curl --help** or **curl --manual** to get basic information about **cURL**.*
While the cURL command line tool used here is useful for a variety of URL manipulations and transfers, it's not written to do everything you may want to do. It simply makes the HTTP request, gets the data, sends data, and retrieves information in a single command line. You could alternately choose to manually make repeated single command line invokes, or you could instead glue everything together inside a unified program framework with a scripting language like JavaScript or another computing language (like Java, C#, or C++)…virtually every programming language has some low level interface to handle an HTTP request-response cycle, their detailed descriptions are too advanced for this introduction.

Getting Started Sending Command Lines to the NT RESTful API Using cURL

1. First Set Up Your CLI Environment for issuing Command Lines:

Use the "Command Prompt" of the host client's Operating System or install an HTTP client tool such as "Postman," or create your own written script for executing curl commands and sending requests to the Acromag NT series device (I/O server).

*The following REST service examples will use **curl**, a command-line tool for transferring data that supports about 22 protocols, including HTTP (the HTTP request could alternately be sent using any other method of sending HTTP requests). A REST API is not always limited to HTTP, but HTTP is the most common protocol used and the one we use here.*

2. **Begin a session by Authenticating yourself with the remote NT system**

   Before you can make any server requests, you need to **log in** to obtain a randomly generated *session ID*. Use the following Login command line to log in to the server and obtain a session ID, as this will be needed for subsequent server requests.

   curl -X POST -d "username=<your_username>" -d "password=<your_password>" http://<IP_AddressOfNT>/webif/login

3. **Retrieve NT Slot Information:**

   Once you are logged in, you may use the GET Slots command to retrieve information about each of the NT server occupied I/O slots: the total number of occupied slots and the type of I/O card installed in each slot (see Table 1).

   curl -X GET http://<IP_AddressOfNT>/webif/slots_sessionID=<your_sessionID>

   Recall that the Acromag Series NT carries its I/O groups in four slots numbered 0-3. Slot 0 always refers to the I/O card internally mated to the system's main CPU module (NTE2xxx or NTW2xxx module). Up to 3 more NTX2xxx boxcar I/O modules may occupy I/O slots 1-3 that connect to the main CPU module along its DIN rail communication bus. Refer to Table 1 and note that there is a choice of 18 unique I/O groups or I/O expansion models that can mix or match together among these four slots.

   Note the I/O of each slot shares a common return among all four slots, making I/O as one group isolated from the CPU network and power circuits.

4. **Explore NT Slot Detail:**

   To retrieve more detailed information about a specific slot, use the following GET Slot X command which returns information about a particular I/O group installed at the selected slot, such as channel type and channel count.

   curl -X GET http://<IP_AddressOfNT>/webif/slots/<SlotNumber>_sessionID=<your_sessionID>

5. **Retrieve Number of Channels**

   To retrieve the number of available channels of a specific channel type at a specific slot, use the following GET Number of Channels of Channel Type Y of Slot X command.

   curl -X GET http://<IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>_sessionID=<your_sessionID>

   Example cURL CLI Command: You may wish to verify the number of available channels of an analog-input channel type for a module installed at slot 2 (i.e. corresponds to the second NTX boxcar of the system). With this command, you may confirm if its second I/O expansion module has 8 analog input channels. The provided example command highlights this with "analog-input" after the "/slots/2/" portion in the command.

   curl -X GET http://<IP_AddressOfNT>/webif/slots/2/analog-input_sessionID=<your_sessionID>

6. **Retrieve Channel Value**

   To monitor or read channel values, or the value of a specific channel, use the Get Value of Channel Z of Channel Type Y at Slot X command. This command helps you access real-time data from the NT devices channel, depending on the type of channel you are querying.

   curl -X GET http://<IP_AddressOfNT>/webif/slots/<SlotNumber>/ <ChannelType>/ <ChannelNumber>_sessionID=<your_sessionID>

   Example cURL CLI Command: To retrieve the current value of a channel, such as its resistance measurement, perhaps the first analog-input channel, you could use this command to retrieve the measured resistance at analog-input channel 1 of the I/O board in slot 2 (the first channel of the 2nd NTX expansion module, an NTX2621 RTD Module).

   curl -X GET http://<IP_AddressOfNT>/webif/slots/2/analog-input/1_sessionID =<your_sessionID

7. **Modify Channel Values**

   To control an output channel by setting its output value at a specific channel, use the Change Value of Channel Z of Channel Type Y of Slot X command. This command allows you to modify the output state of a channel, perhaps by turning a device ON or OFF for tandem digital input/output channels and/or just digital outputs, or for adjusting the output level of analog-output channels.

   curl -X POST -d "newValue=<desired_value>" http://<IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>/ <ChannelNumber>_sessionID=<your_sessionID>

Example cURL CLI Command: This command is useful in many scenarios according to the channel type. Perhaps you wish to control a relay connected to slot 2, channel 1, which is a digital I/O (discrete input/output channel). Running this command can turn the relay ON and feedback the output state to see if the relay is working via its tandem digital input.

curl -X POST -d "newValue=0" http://<IP_AddressOfNT>/webif/slots/2/digital-input-output/1_sessionID=<your_sessionID>

Another scenario might be to change the specific value of an analog-output channel at channel 1 of the I/O expansion module installed in slot 2.

curl -X POST -d "newValue=4.5" http://<IP_AddressOfNT>/webif/slots/2/analog-output/1_sessionID=<your_sessionID>

HTTP Command Requests to the NT RESTful API

The simplified NT RESTful API allows you to communicate with the NT over Ethernet through HTTP requests to read, write, and update NT system I/O resources, using only the HTTP methods of GET to retrieve a record or POST to create/write a new record.

In the command line examples to follow, you will see an NT URL web address combined with an HTTP Method of GET or POST, and the modeled URI pointing to the modeled resource and its data parameters.

Remember, the ***URL*** refers to *Uniform Resource **L**ocator* and specifies the location or web address of the NT I/O system server on the network.

The ***URI*** refers to the *Uniform Resource **I**dentifier*, which can identify any type of resource, not just those on the network and is comprised of a specific unique sequence of characters to distinguish the specific resource of the NT server (its unique address for a piece of information found within the NT REST API).

Login Request (Do This First)

Overview/Description

The Login request is always the first step in command line interaction with the NT RESTful API. This command is required for authentication by presenting valid credentials (the NT's username and password). If authentication is successful, a valid session ID will be returned, which must be appended to subsequent requests or commands.

Command Format for Login Request:

URL: <IP_AddressOfNT>/webif/login
METHOD: POST
URI: /webif/login

Parameters: ***username:*** The username of the user attempting to log in
password: The password for authentication.

Example Request/Response for Login:

Command to Execute: curl -X POST -d "username=<your_username>" -d "password=<your_password>" <IP_AddressOfNT>/webif/login

```
H:\>curl -X POST -d "username=admin" -d "password=password" 192.168.0.10/webif/login
{"username":"admin", "sessionID":135514439, "usersRights":4, "status":"Login sucessful."}
```

"username:" The name of the user that just attempted to login the server.

"sessionID:" A unique identifier for the user's session which is used to authenticate subsequent requests by the client by appending this ID to subsequent requests to maintain the session with the server.

"userRights:" Refers to the access rights & permissions assigned to the logged-in user. A value "4" signifies a specific level of access rights and for the Restful API, and a valid login is always denoted with 4.

"status:" The status field provides a message about the result of the login attempt.

Get Slots Request

Overview/Description

The Get Slots request allows a user to retrieve information about all the slots of an NT series system and its response will return a list of connected slots along with detail as to the type of I/O card installed in each slot. This is important for the user to be able to monitor and/or control the NT device's I/O as it gives information regarding the configuration of the connected module(s).

Command Format of a Get Slots Request

URL: <IP_AddressOfNT>/webif/slots
METHOD: GET
URI: /webif/slots
Parameters: None are required for this request

Example Request/Response of Get Slots

Command to Execute: curl -X GET http://<IP_AddressOfNT>/webif/slots

```
H:\>curl -X GET 192.168.0.10/webif/slots_sessionID=421347657
{"status":1, "numberOfSlots":4, "slotDetails":["Digital I/O Board (Relay)","6-ch 120 VAC Input","Analog Current Out Board","Thermocouple"]}
```

"status:" This field indicates whether the request was successful and a value of 1 denotes a successful response.
"numberOfSlots:" This field shows the total number of slots connected at the NT device.
"slotDetails:" This array provides details about the types of I/O cards installed in each of the slots 1-4 (see Table 2). Each element in the array corresponds to a specific slot (0-3).

Table 2: API Channel Type for NT I/O Model Groups of NTE/NTW and NTX Modules

NT I/O MODEL	RESTful API CHANNEL TYPE	CHANNEL GROUP	CHANNEL GROUP DESCRIPTION (NT systems support 1-4 I/O Groups)
NT2111	digital-input-output	16CH DIO-SINK	16CH DI/O Sinking
NT2121	digital-input-output	16CH DIO-SOURCE	16CH DI/O Sourcing
NT2711	digital-output digital-input-output digital-input	3CH DO-DIFF (CTRL) 4CH DIO-SINK 3CH DI-DIFF (ENCO)	Stepper Motor Controller/Encoder
NT2131*	digital-output digital-input	6CH DO-MR 6CH DI-AHI	6CH Mechanical Relay & 6CH DI Active High
NT2141	ac-input	6CH DI-130VAC 2CH DIO-SINK	6CH 130V AC ISO Input/2CH DI/O Sinking
NT2142	ac-input	6CH DI-240VAC 2CH DIO-SINK	6CH 240V AC ISO Input/2CH DI/O Sinking
NT2211	analog-input digital-input-output	8CH AI-DIFF 2CH DIO-SINK	8CH Differential Current & 2 DI/O Sinking
NT2221	analog-input	16CH AI-SEI	16CH Single-Ended Current Input
NT2231	analog-input digital-input-output	8CH AI-V DIFF 2CH DIO-SINK	8CH Differential Voltage Input & 2 DI/O Sinking
NT2241	analog-input	16CH AI-SEV	16CH Single-Ended Voltage Input
NT2311	analog-output	8CH AO-I (0-22mA)	8CH 0-22mA Current Output
NT2321	analog-output	8CH AO-V w/COM	8CH Voltage Output w/Common Return
NT2511	analog-input digital-input-output analog-output	4CH AI-V DIFF 4CH DIO-SOURCE 2CH AO-I (0-22mA)	4CH Diff I+4CH DI/O SRC+2CH AO-I Combo

*Note: The NT2131 I/O group is not available as an NTW model, only as an NTE CPU or NTX expansion module.

NT I/O MODEL	RESTful API CHANNEL TYPE	CHANNEL GROUP	CHANNEL GROUP DESCRIPTION (NT systems support 1-4 I/O Groups)
NT2531	analog-input digital-input-output analog-output	4CH AI-V DIFF 4CH DIO-SOURCE 2CH AO-I (0-22mA)	4CH Diff V+4CH DI/O SRC+2CH AO-I Combo
NT2512	analog-input digital-input-output	4CH AI-I DIFF 4CH DIO-SOURCE	4CH Diff I+4CH DI/O SRC Combo (no AO-I)
NT2532	analog-input digital-input-output	4CH AI-V DIFF 4CH DIO-SOURCE	4CH Diff V+4CH DI/O SRC Combo (no AO-I)
NT2611	analog-input digital-input-output	8CH AI-mV/TC 2CH DIO-SINK	8CH Thermocouple/mV & 2 DI/O Sinking
NT2621	analog-input digital-input-output	4CH AI-RTD 2CH DIO-SINK	4CH 2/3/4-Wire RTD/Resistance & 2 DI/O Sinking

Get Slot X Request

Overview/Description

The Get Slot X request is used to retrieve detailed information about a specific slot that is connected to the NT series device. This curl command provides information on the channel types present within a given slot, such as digital-input, analog-output, ...etc (see Table 2). This allows the user to determine and understand what data a particular connected slot may be holding.

Command Format of a GET Slot X Request

URL: <IP_AddressOfNT>/webif/slots/<SlotNumber>
METHOD: GET
URI: /webif/slots/<SlotNumber>

Parameters: **SlotNumber**: The specific slot number (0, 1, 2, or 3) to retrieve the info from.

Example Request/Response of Get Slot X:

Command to Execute: curl -X GET http://<IP_AddressOfNT>/webif/slots/<SlotNumber>

```
H:\>curl -X GET 192.168.0.10/webif/slots/1_sessionID=421347657
{"status":1, "singleSlotDetails":[{"channel-type":"digital-input-output", "channels":2}, {"channel-type":"ac-input", "channels":6}]}
```

"status:" This field indicates whether the request was successful. In this case, a value of 1 denotes a successful response.
"singleslotDetails:" This array provides details about the type of I/O card installed in the slot. Each element in the array corresponds to a single specific slot.
"channels:" This field within "singleSlotDetails" provides a more detailed description of the channels available at the requested slot.

Get Number of Channels for Channel Type Y of Slot X

Overview/Description

The Get Number of Channels for Channel Type Y of Slot X request allows a user to retrieve the number of available channels of a specific channel type within a given slot. This curl command can help users understand the configuration of the I/O cards installed in the connected slot, specifically focusing on the number of channels available for the I/O operations.

Command Format for Get Number of Channels of Type Y of Slot X

URL: <IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>
METHOD: GET
URI: /webif/slots/<SlotNumber>/<ChannelType> See Table 2 for Channel Type by I/O

Parameters: **SlotNumber:** The specific slot number (e.g., 0, 1, 2, and 3) to query.
ChannelType: Type of channel in the slot (e.g., digital-input, analog-output), see next page.

Example Request/Response for Get Number of Channels of Type Y of Slot X

Command to Execute: curl -X GET http://<IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>

```
H:\>curl -X GET 192.168.0.10/webif/slots/1/digital-input-output_sessionID=421347657
{"status":1, "amount":2}
```

"status:" This field indicates whether the request was successful. In this case, a value of "1" represents a successful response.
"amount:" This field represents the total number of channels available for the specified channel type in the given slot.

Get Value of Channel Z of Channel Type Y of Slot X

Overview/Description

The Get Value of Channel Z of Channel Type Y of Slot X request allows users to retrieve the number of available channels for a specific channel type within a given connected slot. This curl command can help users understand the configuration of the I/O cards installed in the connected slot, specifically focusing on the number of channels available for the I/O operation.

Command Format of Get Value of Channel Z of Channel Type Y of Slot X

URL: <IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>/<ChannelNumber>
Method: GET
URI: /webif/slots/<SlotNumber>/<ChannelType>/<ChannelNumber>

Parameters:
SlotNumber: The specific slot number (e.g., 0, 1, 2, and 3) to query.
ChannelType: The type of channel in the slot (e.g., digital-input, analog-output).
ChannelNumber: The number of the channel being queried

Example Request/Response for Get Value of Channel Z of Type Y of Slot X

Command to Execute: curl -X GET (See Table 2 for Channel Type by I/O)
http://<IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>/<ChannelNumber

```
H:\>curl -X GET 192.168.0.10/webif/slots/1/analog-input/1_sessionID=311309446
{"status":1, "channelNumber": 1, "analogInputValue": 537.400, "units": "Ω"}
```

"status:" This field indicates whether the request was successful. In this case, a value of "1" represents a successful response.
"channelNumber:" This field represents the channel number being queried. In this case, the requested channel is Channel 1.
"analogInputValue:" This field provides the current value of the requested channel. Here, the analog input value for Channel 1 is 537.400.
"units:" This field specifies the units of measurement for the analog input value of this specific channel. In this example, the value measured is resistance in ohms (Ω).

Change Value of Channel Z of Channel Type Y of Slot X

Overview/Description

The Change Value of Channel Z of Channel Type Y of Slot X request allows users to modify the output value of a specific channel of the specific slot. This command is useful for controlling the output channels of an NT series I/O system, such as setting digital outputs on/off, or adjusting analog output levels.

Command Format for Change Value of Channel Z of Channel Type Y of Slot X

URL: <IP_AddressOfNT>/webif/slots/<SlotNumber>/<ChannelType>/<ChannelNumber>
Method: POST
URI: /webif/slots/<SlotNumber>/<ChannelType>/<ChannelNumber>

Parameters:

SlotNumber: The specific slot number (e.g., 0, 1, 2, and 3) to query.
ChannelType: The type of channel in the slot (e.g., digital-input, analog-output).
ChannelNumber: The number of the channel being queried
newValue: The value to be set for the specified channel (e.g. use a "1" to set the digital-input-output channel ON, or "5.0" to change the output level an analog output channel to 5V or 5mA).

Example Request/Response for Change Value of Channel Z of Channel Type Y of Slot X

Command to Execute: curl -X POST -d " value=<your_value>" http://<IP_AddressOfNT>/webif/ slots/<SlotNumber>/<ChannelType>/<ChannelNumber> (See Table 2 for Channel Type by I/O)

```
H:\>curl -X POST -d "newValue=1" 192.168.0.10/webif/slots/1/digital-input-output/1_sessionID=365732339
{"status":1, "message":"Value was successfully changed."}
```

```
H:\>curl -X POST -d "newValue=5.0" 192.168.0.10/webif/slots/1/analog-output/1_sessionID=365732339
{"status":1, "message":"Value was successfully changed."}
```

"status:" This field indicates whether the request was successful. For both examples, a value of 1 represents a successful response.

"message:" The message for the first case above confirms that the value of the specified digital-input-output channel has been successfully updated to the desired value (1 in this case, which typically means "ON").

The message for the second case above confirms that the value of the specified analog output channel has been successfully updated to the new value (5.0 for this example).

Preventing Error Messages When Using cURL to Make Requests to an NT Server

Sometimes, when working with the various RESTful API cURL commands to interact with a remote NT device, errors may occur if a command is not properly formatted, or certain conditions are not met. Below are some general guidelines for a variety of situations to help avoid some common errors with each type of request.

1. **Provide a Valid Session ID**

   Ensure that you always include a valid sessionID with each request which you obtain after your successful login attempt with the Login command (additional commands require this session ID for authentication and to keep the session open).

   If the session ID is missing or expired for your command, you will get an authentication error and will have to log in again to retrieve a new sessionID.

2. **Use the Correct HTTP Method**

   Ensure that you are using the correct HTTP method for the command you are using. The GET method is used for retrieving data (e.g, slot details, channel values, etc), while the POST method is used for modifying data (e.g. like changing channel values).

   Depending on the command, use of POST instead of GET, or GET instead of POST, will result in an error.

3. **Avoid Extra Characters or Typos in the URL**

   Ensure that the URL for any cURL command is typed correctly, with no extra characters, missing parts, or misspellings.

   Always double-check the format of the URL and use the exact format depicted for the command. Any typo will result in errors.

4. **Provide Correct Credentials**

   Ensure the spelling of username and password matches the credentials stored in the NT series device. Even a minor typo will result in an "Invalid Username/Password" error.

   Be sure to provide valid/precise username & password when making the login request

5. **Include All Required Parameters**

   Always include **-d "username=<your_username>" -d "password=<your_password>"** in the curl command to ensure all required information is provided. Furthermore, they must be in this order to prevent an error from being returned.

   The login request requires both the username and password parameters. If either of these parameters are missing or out of order, an error will be returned.

6. **Use a Valid Slot Number**

   Ensure that you input a valid SlotNumber at a particular Series NT system device (valid meaning occupied). The valid slot number range is 0-3 (integer only).

   Use of an invalid/unoccupied slot number will return an "Invalid Slot Number" error.

7. **Use a Valid Channel Type**

   Ensure that the ChannelType specified in the URL is valid and exists for your NT series I/O device.

   Check the spelling of the ChannelType before using this command. In relation to the example usage, common errors, such as typos like "anog-inut" instead of "analog-input", will lead to an error. Refer to the list of valid channel types under the "Valid Channel Types and Associated Valid Board Types" section.

8. **Verify Channel-Slot Compatibility**

   Ensure that the channel type specified in the URL is valid and associated with the correct I/O board that occupies the specific slot.

   Use the Get Slot X command to check which channel types are available in a specific slot. Only use a compatible channel type to avoid errors.

9. **Use a Valid Channel Number**

   Ensure that the ChannelNumber specified in the URL is valid and exists for your NT series device. Using an invalid or out-of-range channel number will lead to an error.

   Verify the total number of channels available using the Get Number of Channels for Channel Type Y of Slot X curl command. Make sure that the requested Channel Number is within a valid range.

10. **Use a Valid Channel Type and Slot Number**

    Ensure that the ChannelType and SlotNumber specified are valid. Using an incorrect channel type or slot number will lead to an error indicating that the request cannot be completed.

    First, use Get Slots and Get Slot X commands to determine the available slots and channel types for each slot. This will help you select the correct ChannelType and SlotNumber before making the request.

11. **Set the Appropriate Value for Channel Type**

    Digital-Input-Output: Use either 0 (for OFF) or 1 (for ON) to toggle and set the value of the specified discrete channel.

    Setting any value other than 0 or 1 will cause an error, as the digital-input-output channels can only accept these specific set values.

    Digital-Output: Use 0 (for OFF) or 1 (for ON) to toggle and set the value of the specified discrete output.

    Setting any value other than 0 or 1 will cause an error, as the digital-output channels can only accept these specific set values.

    Analog-Output: Use a numeric value appropriate for the range and units defined for that channel.

    Use the Get Slot X command to determine the acceptable range and units for the analog output channel to avoid setting a value that is out of bounds.

12. **Ensure Correct Curl Command Format**

    Ensure that the -d parameter is used to specify the value that needs to be set for the channel. It must be written as -d "value=<your_value>".

    Avoid leaving out the -d parameter or misspelling it. An incorrect or missing parameter will result in an error.

13. **Avoid Unintended Characters (#,?,^, etc.) in Any API Request you make.**

    The format of your request must ***exactly*** match the Request Model provided for the NT RESTful API request you are making. Any substitution of characters, special characters, or use of the opposite case may cause an unexpected error response.

    Follow the API Request models exactly as noted when making your request— do not substitute or omit characters or reverse the character case to avoid errors.

14. **Do NOT issue cURL commands to a slot that is not connected for Any API Request you make.**

    If a curl command is executed on a slot that is not connected, the error message "This slot is not in use" will be returned.

    Ensure a Slot is Connected before issuing an API Request. Before interacting with a slot, use the Get Slots command to verify which slots are currently in use. Attempting to retrieve or modify data from an unused slot will result in an error.