Technical Document

Niagara HTTP Client Driver Guide

24 November 2021

niagara4

Niagara HTTP Client Driver Guide

Tridium, Inc.
3951 Westerre Parkway, Suite 350
Richmond, Virginia 23233
U.S.A.

Contents

About this guide

This topic contains important information about the purpose, content, context, and intended audience for this document.

Product Documentation

This document is part of the Niagara technical documentation library. Released versions of Niagara software include a complete collection of technical information that is provided in both online help and PDF format. The information in this document is written primarily for Systems Integrators. To make the most of the information in this book, readers should have some training or previous experience with Niagara software, as well as experience working with JACE network controllers.

Document Content

The HTTPClient driver provides the tools to connect Niagara with HTTP services, such as web services and restful API endpoints. This facilitates the exchange of data both in and out of a Niagara station.

Document change log

Changes to this document are listed in this topic.

November 24, 2021

- Added Http Client Service.
- Added prerequisites to several requests.
- Added two bullet points to the Security dashboard.
- Added additional properties to Http Client Service, Http Tuning Policy, Request Throttle and Client Request History

October 5, 2021

Initial document release.

Related documentation

Additional information is available in the following documents.

- *Getting Started with Niagara*
- *Niagara Drivers Guide*

Chapter 1 Introduction

Topics covered in this chapter

♦ Feature summary
♦ Licensing
♦ Palette and modules

The HTTP Client module provides tools and a driver, which interact with HTTP services, such as web services and restful API endpoints. This transport permits data exchange both in and out of a station.

HTTP Clients provide the functionality to execute a `GET`, `POST` or `PUT` command between Niagara and compatible web services and APIs.

An API (Application Programming Interface) allows two applications to interact with each other. This may be a local IoT device or an external web service or web page. Examples include:

- A REST API that supplies external data, such as weather forecasting, live travel times, local air quality etc.
- A web service that populates an external data source, such as a database with building and sensor data.
- Local devices that expose an API to control or monitor functionality.

API's support many data formats. The predominant use of JSON in modern web services allows easy integration between this client tool and the JSON Toolkit module.

Feature summary

The HTTP Client driver supports features designed to make configuration easy and intuitive.

- A standalone HTTP Client component that supports individual HTTP requests.
- HTTP device and proxy extensions that support multiple related points, which are required to send requests in a regular, predictable manner
- User configurable headers and parameters with auto-complete on names
- Auto headers for some values (Host, Content-Type, Date)
- Multiple methods of authentication: Http Basic, Http Digest, Bearer Token, Niagara SCRAM-SHA, Cookies
- Choice between standard Java or OKHttp library connection transport layer
- Response headers with cookie capture
- Request POST and PUT body, which may be a string, file or report
- Standard Niagara tuning options
- Http-specific options, such as follow redirects and use caches
- Ability to quickly duplicate many copies of an http client or proxy extension with changes
- Ability to populate a client's address and parameters by pasting in a url address
- Metrics on request and response statistics
- Ability to trigger secondary requests based on the outcome of a prior request
- Security dashboard cards
- WebSocket Clients Component

Licensing

A license and SMA are required to use this driver.

Client license

To use the HTTP Client, your host requires the 'http' feature added to the host's license. Production (non-demo) licenses also require an active SMA (Software Maintenance Agreement) for the module to function. Engineering or Demo licenses should have this feature added by default.

Capacity Licensing

The standalone HTTP client counts as one (1) point in global capacity. Driver points count as one (1) proxy point each as per other Niagara drivers.

SMA Expiration Monitor

In addition to the license requirement, the module requires an active SMA. The Expiration Monitor increases notifications as expiration of this agreement approaches. It runs on startup. The monitor (of the HttpClient-Service) checks every 24 hours to establish if the expiration date is within the warning period, or expired, and generates an offNormal or fault alarm accordingly. Although the alarms are likely the most accessible type of notification, the SMA Expiration Monitor also logs the days remaining to the station console, which, for example, could be shown on a dashboard. The station's **UserService** has an `SMA Notification` property that alerts users when they log in.

As the extension of the SMA currently requires a reboot to install the new license, once the monitor detects that the agreement has expired, it performs no further checks until the station starts again.

Palette and modules

A single palette and three core Niagara modules support this driver.

The palette is `httpClient`.

The four modules are:

- httpClient-rt
- httpClient-ux
- httpClient-wb

Chapter 2 Setup

Topics covered in this chapter

Basic driver set up involves adding an **HttpClient** and/or **HttpClientNetwork**, devices and points to the station. These components function as standard Niagara driver components.

After setting up the basic components you configure HTTP requests and responses.

Setting up the Http Client Service

The **HttpClientService** allows the use of all Http Client types within the station.

Prerequisites: The module is licensed and has an active SMA. You are working in Workbench running on a PC or laptop and are connected to a station.

Step 1 Open the `httpClient` palette.

Step 2 Drag an **HttpClientService** component to the **Services** folder in the station.

The **Http Client Service Property Sheet** opens.

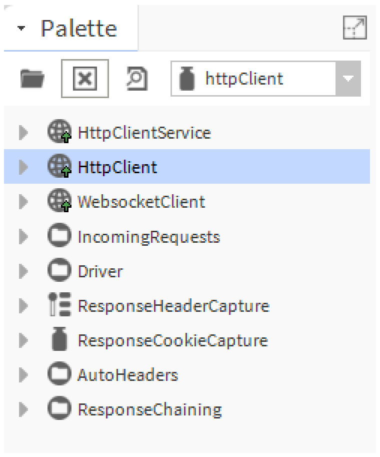
Step 3 Optionally, set the **`enableNonDriverClients`** property to `true`.

This is required for all standalone client types.

Client types

The **`httpClient`** module provides two options for creating clients: a standalone **HttpClient** component and a multple-endpoint **HttpClientNetwork** component.

Figure 1 Client types



Standalone HttpClient

You may use this standalone component to make individual connections to single endpoints using any type of request (GET/POST/PUT) with several configurations, such as parameters, headers and message body. A user invocation or an input into the **Send** action slot triggers the **HttpClient** component's **Send** action.

NOTE: You must enable the Standalone client type in the **HttpClientService** prior to use.

HttpClientNetwork

This component offers the same functionality as the standalone client, but allows several related endpoints to exist as child StringPoint components with configurable proxy extensions per request. Each request can have a different address or a different set of parameters, headers and message body. As the points are part of the standard Niagara driver model, the driver polls these string components according to its tuning policy.

Other benefits include:

- Writable points with priority levels
- The ability to add history, alarm ad other extensions
- Manager views
- Optional device ping to indicate service health

Adding an HttpClientNetwork and device

The **HttpClientNetwork** and **HttpClientDevice** set up the HTTP Client driver in a station.

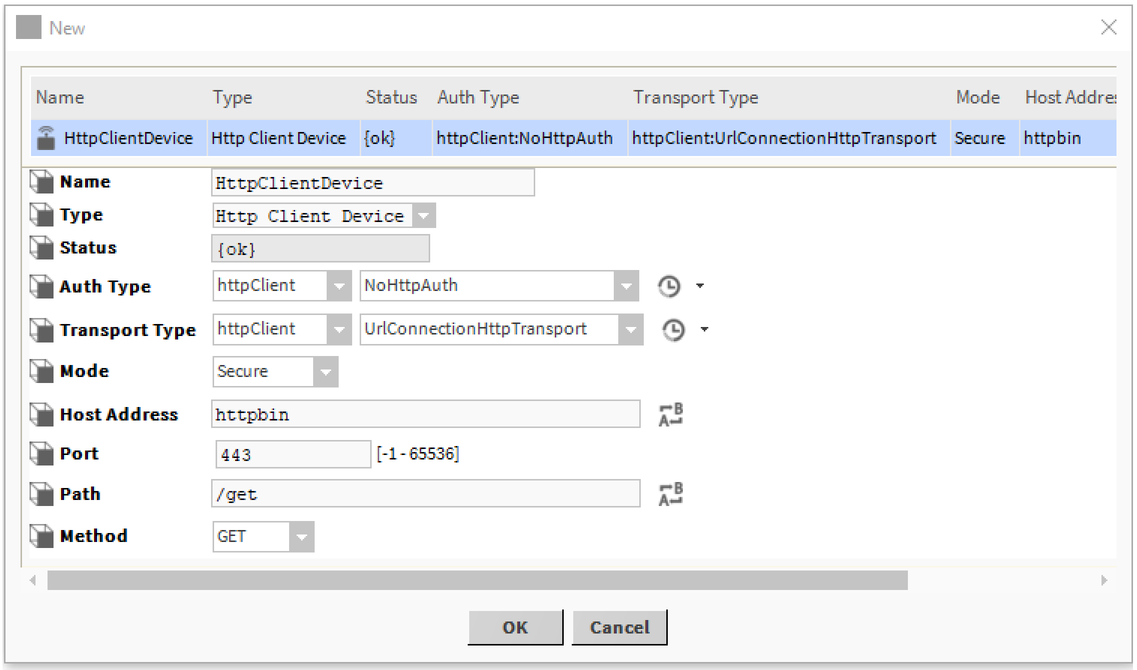
Prerequisites: You are connected to a station. The `httpClient` palette is open. **HttpClientService** has been added to the station's **Services** container. Standalone client types in the **HttpClientService** have been enabled.

Step 1 Expand the **Driver** folder in the palette and drag an **HttpClientNetwork** component to the **Config→Drivers** folder in the station.

Step 2 Do one of the following:

- Drag an **HttpClientDevice** from the palette to the network component you just added to the station and double-click the device component.
- Double-click **HttpClientNetwork**, click **New** and click **OK**.

The **Add** or **Edit** device window opens.



New

Name	Type	Status	Auth Type	Transport Type	Mode	Host Addre
HttpClientDevice	Http Client Device	{ok}	httpClient:NoHttpAuth	httpClient:UrlConnectionHttpTransport	Secure	httpbin

Name	HttpClientDevice
Type	Http Client Device
Status	{ok}
Auth Type	httpClient NoHttpAuth
Transport Type	httpClient UrlConnectionHttpTransport
Mode	Secure
Host Address	httpbin
Port	443 [-1 - 65536]
Path	/get
Method	GET

You may optionally specify an address for a given device that sends data based on the device's Ping Monitor. This regularly polls an address, which indicates the up status of the web service or physical device.

Step 3 Populate at least **Host Address**, **Port**, **Path** and **Method** and click **OK**.

Configuring **Host Address** sets the **Address** property under **Ping Address**. The driver pings this address based on the network's **Ping Monitor** settings. The **HttpClientDevice** also has a **Ping** action.

If you defined a **Ping Address** for the device, and the driver either fails to contact that address or the device receives an unsuccessful response code (non 200), as usual the device points will be affected by the overall health of the device.

Adding points

Points require configuration and parameters.

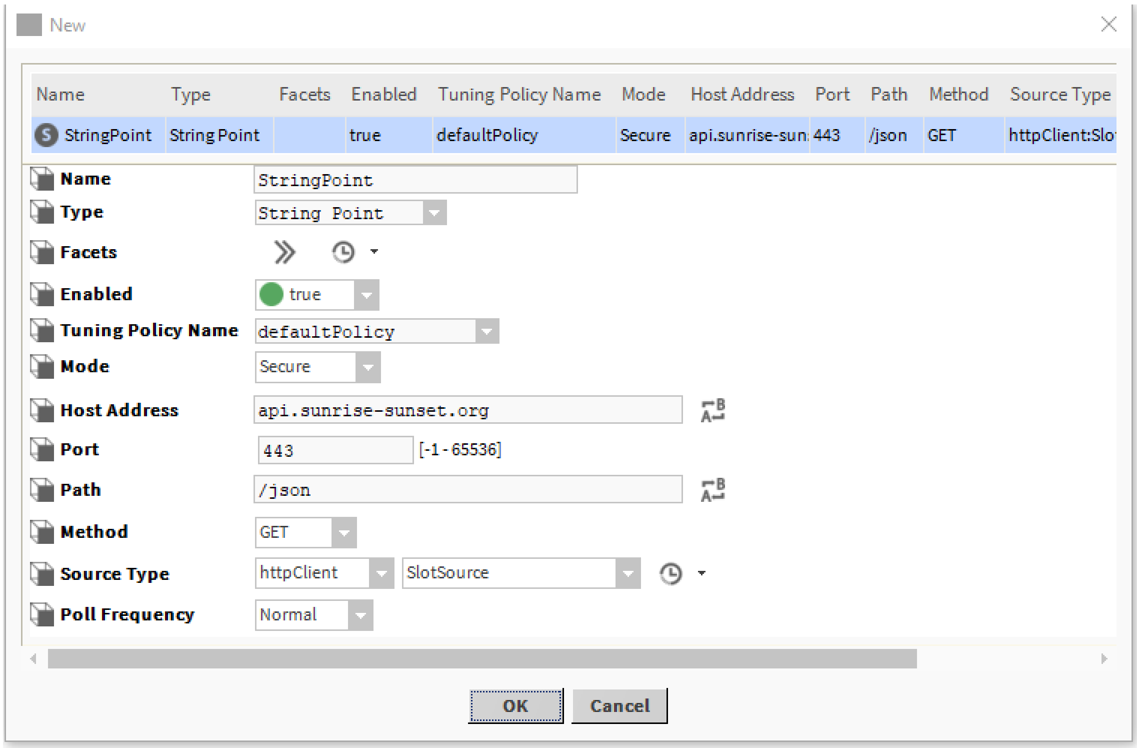
Prerequisites: You are connected to a station.

Step 1 Expand **Config→Drivers→HttpClientNetwork→HttpClientDevice** and double-click **Points**.

The **Http Client Point Manager** opens.

Step 2 To add one or more points, click **New**.

The **New** point window opens.



Name	Type	Facets	Enabled	Tuning Policy Name	Mode	Host Address	Port	Path	Method	Source Type
StringPoint	String Point		true	defaultPolicy	Secure	api.sunrise-sun	443	/json	GET	httpClient:Slo

Name	StringPoint
Type	String Point
Facets	
Enabled	true
Tuning Policy Name	defaultPolicy
Mode	Secure
Host Address	api.sunrise-sunset.org
Port	443 [-1 - 65536]
Path	/json
Method	GET
Source Type	httpClient SlotSource
Poll Frequency	Normal

Step 3 Populate the address of the endpoint including **Host Address**, **Port**, **Path** and **Method**.

Step 4 Set **Poll Frequency** for each point depending on how often each point requires polling and click **OK**.

NOTE: Some services may throttle the number of requests in a given timeframe and/or may charge according to the number of requests. Diagnose an intermittent `{fault}` status on a point using the HTTP Response's **Health** properties. The default poll frequencies are:

- Fast: 5 seconds
- Normal: 5 minutes
- Slow: 15 minutes

You can modify these defaults in the **Poll Scheduler** container within the **HttpClientNetwork Property Sheet**.

For this API, the latitude (lat) and longitude (lng) parameters are required to specify the location of the data point.

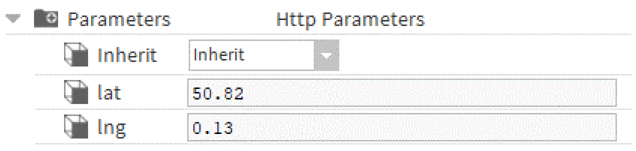
Step 5 To define latitude (lat) and longitude (lng) parameters, double-click the point you just added, expand **Proxy ExtParameters**, right-click **Parameters** and click **Actions→Add**.

The **Add** window opens.

Step 6 For **Slot Name**, enter `lat` and click **OK**.

Step 7 Do the same to enter `lng` and click **OK**.

The latitude and longitude parameter properties open.

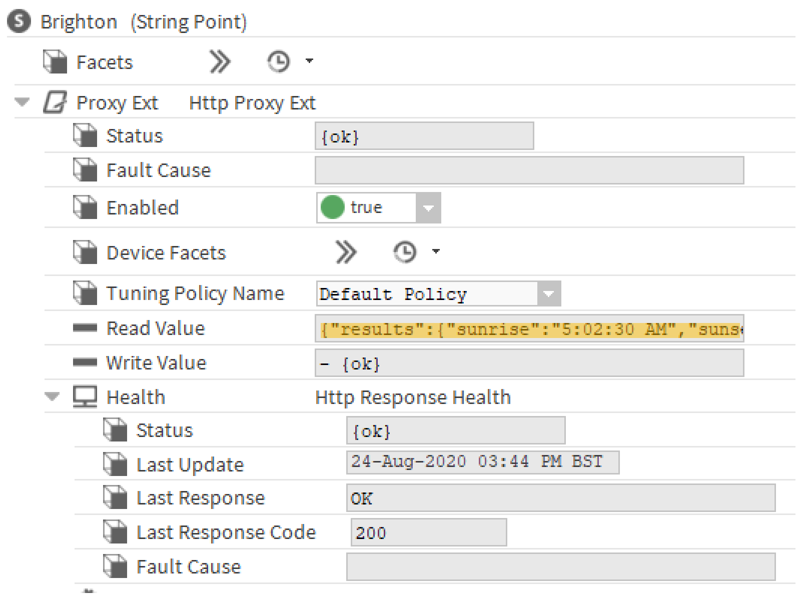


Parameters	Http Parameters
Inherit	Inherit
lat	50.82
lng	0.13

Step 8 Enter the latitude and longitude values and click **Save**.

The **Http Client Proxy Ext** contains all of the features of the standalone client including **`Method`** (GET/POST/PUT), **`Health`**, authentication, **`Request Body`**, **`Parameters`** and **`Headers`**.

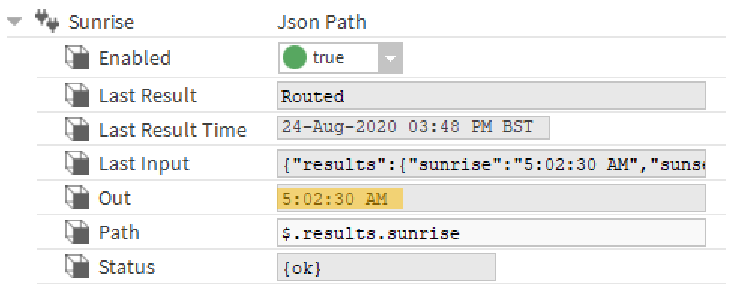
This driver point sends the HTTP request when subscribed and per its selected poll rate in the Poll Scheduler. The **Proxy Ext** includes a **Send** action, which you can trigger if required.



Brighton (String Point)	
Facets	
Proxy Ext	Http Proxy Ext
Status	{ok}
Fault Cause	
Enabled	true
Device Facets	
Tuning Policy Name	Default Policy
Read Value	{"results":{"sunrise":"5:02:30 AM","suns
Write Value	- {ok}
Health	Http Response Health
Status	{ok}
Last Update	24-Aug-2020 03:44 PM BST
Last Response	OK
Last Response Code	200
Fault Cause	

In the example above, the point's **`Out`** slot or the Proxy Ext's **`Read Value`** slot contains the response body. You may link this value to **Wire Sheet** logic. For example, in the case of a JSON response, a JSON Toolkit component may extract the values and use them.

Using a JSONPath component, you can extract the sunrise/sunset time as a time value.



Sunrise	Json Path
Enabled	true
Last Result	Routed
Last Result Time	24-Aug-2020 03:48 PM BST
Last Input	{"results":{"sunrise":"5:02:30 AM","suns
Out	5:02:30 AM
Path	$.results.sunrise
Status	{ok}

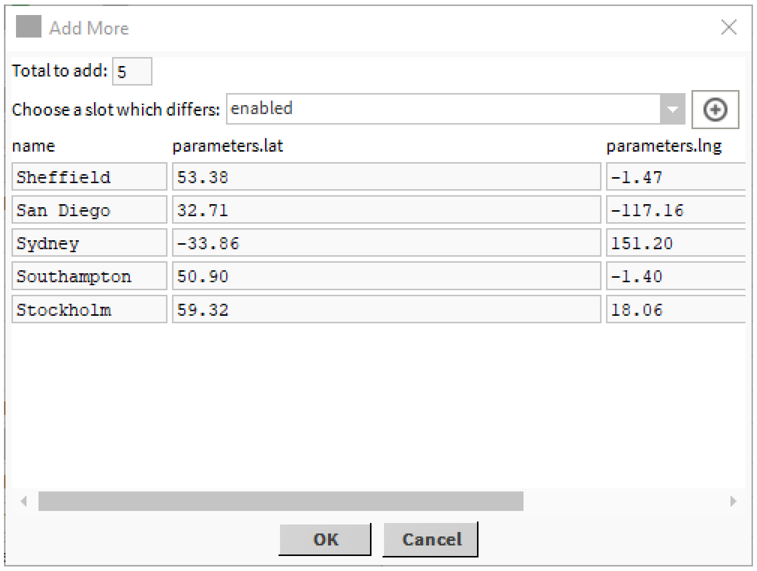
Adding multiple clients and points

The standalone client and http proxy extension both have an **Add More** action that supports copying the current client many times and make changes to slot values.

Step 1 Add a single standalone client or driver point.

Step 2 Expand the client, **Points** folder and point, right-click the **Proxy Ext** and click **Actions→Add More**.

The **Add More** window opens.



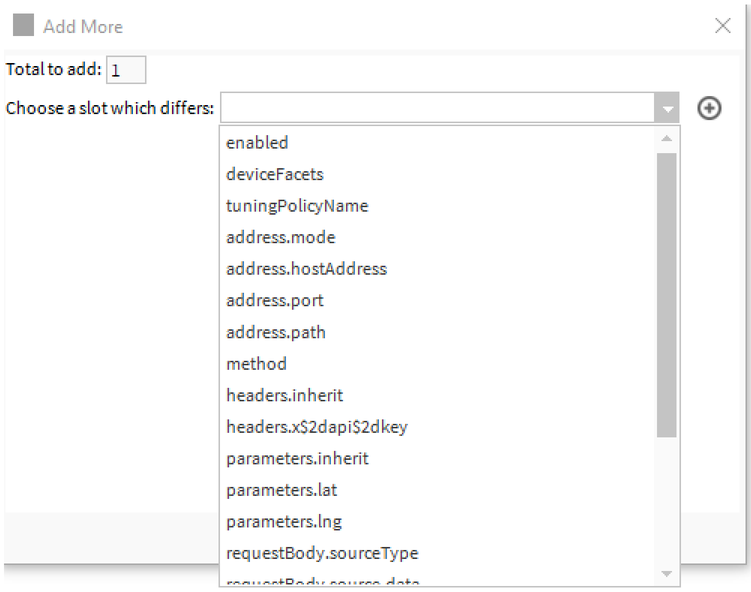
Total to add: 5

Choose a slot which differs: enabled

name	parameters.lat	parameters.lng
Sheffield	53.38	-1.47
San Diego	32.71	-117.16
Sydney	-33.86	151.20
Southampton	50.90	-1.40
Stockholm	59.32	18.06

You may define which slots are to be modified from the original and add several more points without repeating the full configuration of each client and point. The example only requires the name and lat/lng parameters to be changed while adding five new clients.

The **`Choose a slot which differs`** drop-down list offers all non read-only slots from the original client and point:



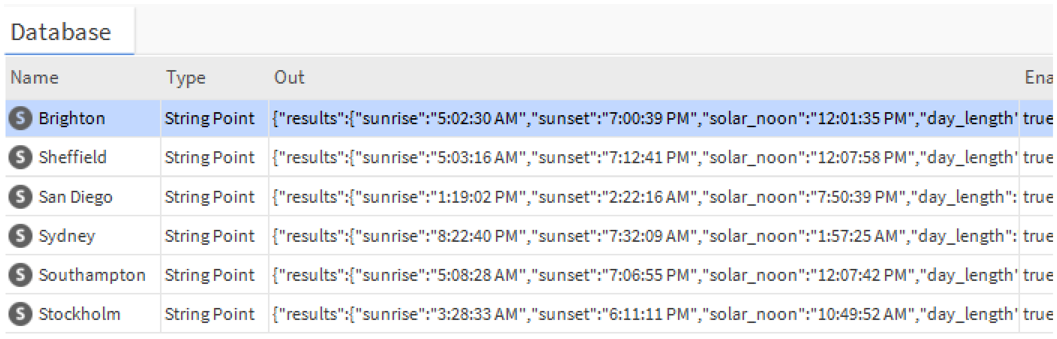
Total to add: 1

Choose a slot which differs:

- enabled
- deviceFacets
- tuningPolicyName
- address.mode
- address.hostAddress
- address.port
- address.path
- method
- headers.inherit
- headers.x$2dapi$2dkey
- parameters.inherit
- parameters.lat
- parameters.lng
- requestBody.sourceType

Step 3 To continue, click **OK**.

The driver adds the new points.



Database

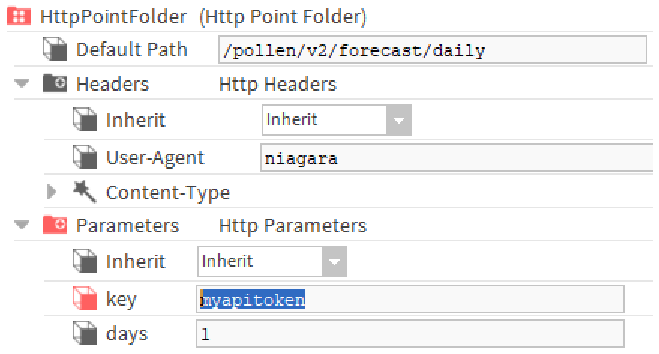
Name	Type	Out	Ena
Brighton	String Point	{"results":{"sunrise":"5:02:30 AM","sunset":"7:00:39 PM","solar_noon":"12:01:35 PM","day_length'	true
Sheffield	String Point	{"results":{"sunrise":"5:03:16 AM","sunset":"7:12:41 PM","solar_noon":"12:07:58 PM","day_length'	true
San Diego	String Point	{"results":{"sunrise":"1:19:02 PM","sunset":"2:22:16 AM","solar_noon":"7:50:39 PM","day_length":	true
Sydney	String Point	{"results":{"sunrise":"8:22:40 PM","sunset":"7:32:09 AM","solar_noon":"1:57:25 AM","day_length":	true
Southampton	String Point	{"results":{"sunrise":"5:08:28 AM","sunset":"7:06:55 PM","solar_noon":"12:07:42 PM","day_length'	true
Stockholm	String Point	{"results":{"sunrise":"3:28:33 AM","sunset":"6:11:11 PM","solar_noon":"10:49:52 AM","day_length'	true

Http Point folder

Http point folders are available in the **Http Client Point Manager**. With a Point folder you specify a default address path, parameters, or headers for all child points in the folder. The default path applies to any child points that do not already have a path, allowing the leading part of the URL to be entered by default.

The driver combines the parameters and headers with those specified on the child points where the child uses the **`Inherit`** setting within the header and parameter folder. The values in the folder take priority in the case of duplicates. For example:

Figure 2 Http Point Folder example



HttpPointFolder (Http Point Folder)

Default Path	/pollen/v2/forecast/daily
Headers	Http Headers
Inherit	Inherit
User-Agent	niagara
Content-Type	
Parameters	Http Parameters
Inherit	Inherit
key	myapitoken
days	1

This example talks to an API for pollen data using the same URL path, api 'key' and 'days' forecast parameter for all child points. All default child points inherit these headers, parameters, and default path. This means they can be defined just once.

The folder also contains a convenience action, **Poll All**, which triggers a send on all child points.

Transport layers

Both the Standalone **HttpClient** and the driver (**HttpClientDevice**) contain a **`Transport Type`** property, which lets you switch the underlying transport layer between that which comes with the standard JRE and the third-party OKHttp library.

This allows the module to potentially work around behaviours seen with either implementation by providing a choice. You may also write your own transport layer in a module and use this instead.

Chapter 3 Requests and responses

Topics covered in this chapter

- ♦ Setting up a GET request
- ♦ Adding parameters
- ♦ Adding headers
- ♦ Setting up a POST request
- ♦ Setting up an AutoHeader
- ♦ Posting file content
- ♦ Posting reports
- ♦ Posting from data
- ♦ Setting up a PUT request
- ♦ Chaining client requests
- ♦ Configuring to fire secondary components
- ♦ Sending and receiving messages with the WebSocketClient
- ♦ Capturing incoming request messages
- ♦ Defining the StringServlet response
- ♦ Monitoring request and response metrics
- ♦ Capturing Response Headers
- ♦ Capturing cookies
- ♦ Troubleshooting

The HTTP Client Driver supports three types of requests: GET, POST and PUT. Responses return information to the source of the request.

The GET, POST and PUT commands retrieve and update data.

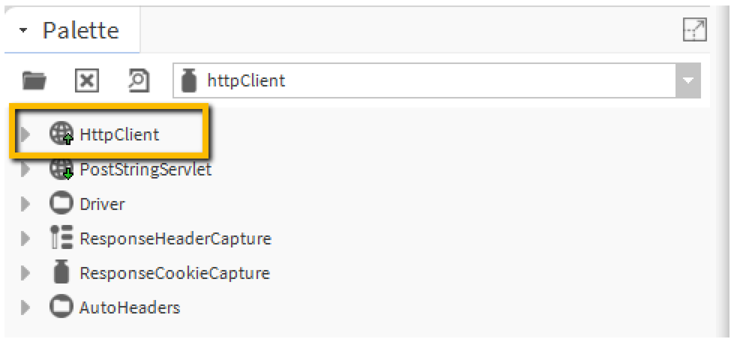
Setting up a GET request

An **HttpClient** component GET is a request to a given endpoint and is often used to retrieve data for a specific resource. It may include various parameters specific to that request.

Prerequisites: You are connected to the station. **HttpClientService** has been added to station's **Service** container. Standalone client types in the **HttpClientService** have been enabled.

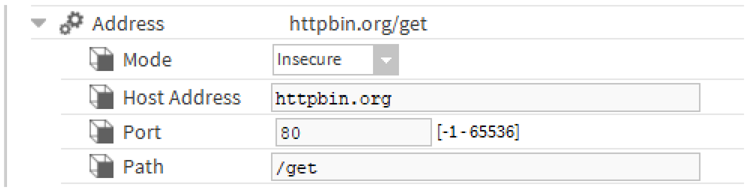
Step 1 Open the `httpClient` palette.

The palette opens.



Step 2 Add an **HttpClient** component to your station from the palette and expand the `Address` slot.

The `Address` properties open.

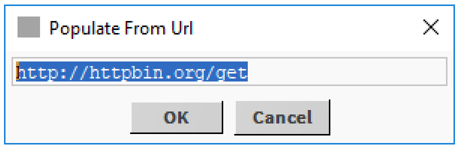


Address	httpbin.org/get
Mode	Insecure
Host Address	httpbin.org
Port	80 [-1 - 65536]
Path	/get

The `Insecure` option for **Mode** configures the **HttpClient** without communication security (TLS, Transport Layer Security) and assumes port 80 by default. The `Secure` option refers to https on port 443 by default.

Step 3 Do one of the following:

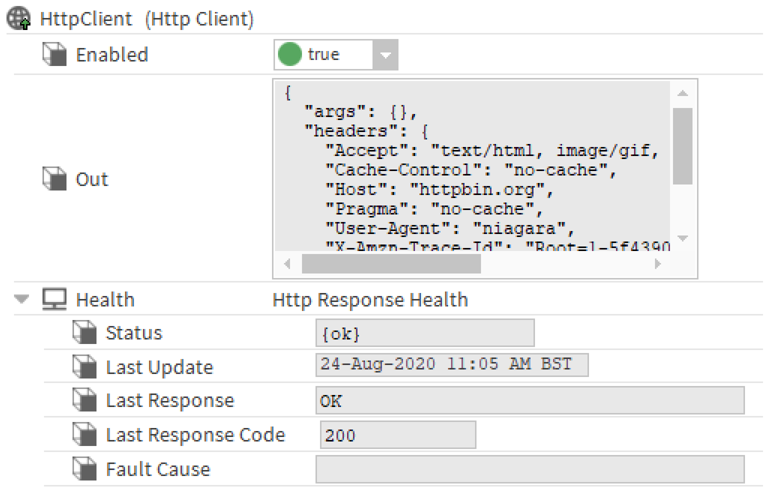
- Populate the **`Mode`**, **`Host Address`**, **`Port`** and **`Path`** properties and click **Save**.
- Right-click **`Address`**, click **Actions→Populate from Url**, paste a complete url in the field and click **OK**.



```
http://httpbin.org/get
```

Step 4 Right-click **HttpClient** and click **Actions→Send**.

The driver makes the request, populates the **`Out`** slot with the http response body and displays the response code and any errors under the **`Health`** slot.



HttpClient (Http Client)

Enabled: true

Out:

```
{
  "args": {},
  "headers": {
    "Accept": "text/html, image/gif,
    "Cache-Control": "no-cache",
    "Host": "httpbin.org",
    "Pragma": "no-cache",
    "User-Agent": "niagara",
    "X-Amzn-Trace-Id": "Root=1-5f4390
```

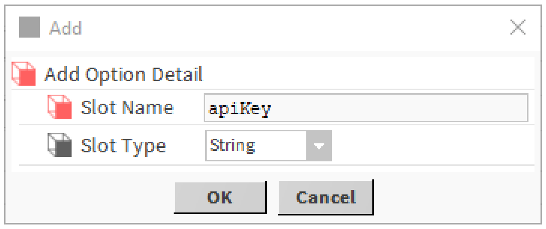
Health	Http Response Health
Status	{ok}
Last Update	24-Aug-2020 11:05 AM BST
Last Response	OK
Last Response Code	200
Fault Cause	

Adding parameters

You may need to define parameter(s) to refine a query request, define an access key or specify an output format. Parameters appear at the end of a url in the form `http://httpbin.org/get?apiKey=5abc7d6cff76==a`

Step 1 Expand **HttpClient** in the Nav tree, right-click on the **Parameters** slot and click **Actions→Add**.

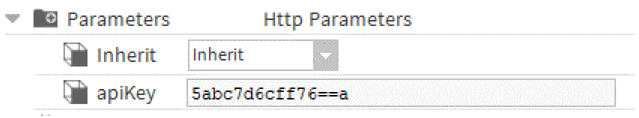
The **Add** slot window opens.



Step 2 Give the parameter a **Slot Name** and click **OK**.

Step 3 Double-click **Parameters**.

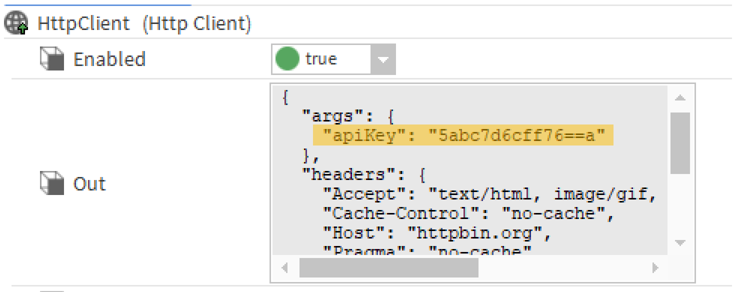
The property you created appears.



Step 4 Enter a value or link to a slot elsewhere in the station to supply the value.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

This API echos any supplied arguments back in the response body.



```
{
  "args": {
    "apiKey": "5abc7d6cff76==a"
  },
  "headers": {
    "Accept": "text/html, image/gif,
    "Cache-Control": "no-cache",
    "Host": "httpbin.org",
    "Pragma": "no-cache"
```

If **Inherit** is set to `Inherit`, the driver merges the **Parameter** values defined within parent components, such as the **HttpClientFolder**, with the child component parameters.

Adding headers

Headers are used to define an access key, to specify your requests content type or acceptable response content types. Unlike parameters, HTTP headers are not part of the address url.

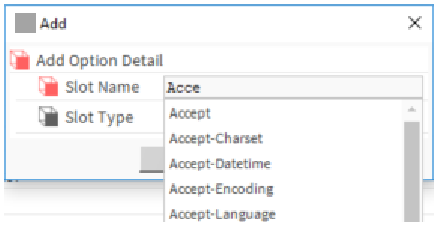
The **HttpClient** automatically sends some headers. The response from `httpbin` echoes back the sent headers:

```
"headers": {
  "Accept": "text/html, image/gif,
  "Cache-Control": "no-cache",
  "Host": "httpbin.org",
  "Pragma": "no-cache",
  "User-Agent": "niagara",
```

You may define your own headers or override the defaults.

Step 1 Expand **HttpClient**, right-click **Headers** and click **Actions→Add**

The **Add** window opens.



Slot Name: Acce

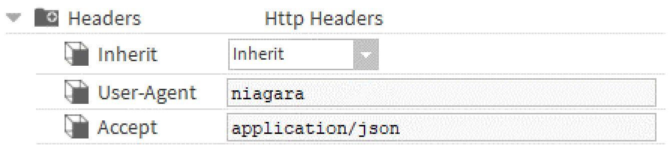
```
Accept
Accept-Charset
Accept-Datetime
Accept-Encoding
Accept-Language
```

Step 2 Start typing a header **Slot Name**.

A drop-down list opens with credentials, headers and methods.

Step 3 Select a header and click **OK**.

The **HttpClient** adds the header under the **Headers** folder.

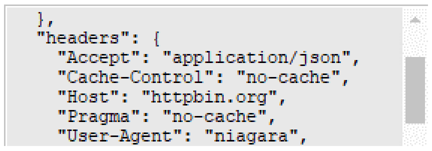


Headers	Http Headers
Inherit	Inherit
User-Agent	niagara
Accept	application/json

Step 4 Manually enter a value or link to a slot elsewhere in the station to supply the value.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

The header has been overwritten.



```
},
"headers": {
  "Accept": "application/json",
  "Cache-Control": "no-cache",
  "Host": "httpbin.org",
  "Pragma": "no-cache",
  "User-Agent": "niagara",
```

If **Inherit** is set to `Inherit`, the driver merges the header values defined within parent components, such as the **HttpClientFolder**, with the child component headers.

Setting up a POST request

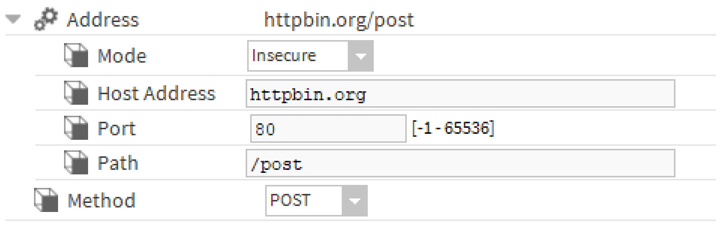
An Http POST request is primarily the same in function as a GET request with the addition of a message body to request or update data within a resource.

Prerequisites: You are connected to the station, which has an HttpClient component. **HttpClientService** has been added to station's **Service** container. Standalone client types in the **HttpClientService** have been enabled.

Step 1 Expand **HttpClient** and change **Method** to `POST`.

Step 2 Expand **Address**, enter the **Host Address** and **Path** and click **Save**.

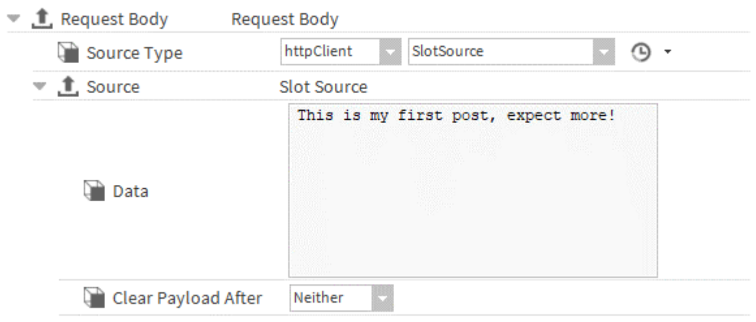
The Address properties are configured for a POST request.



Address	httpbin.org/post
Mode	Insecure
Host Address	httpbin.org
Port	80 [-1 - 65536]
Path	/post
Method	POST

Step 3 Expand **Request Body**.

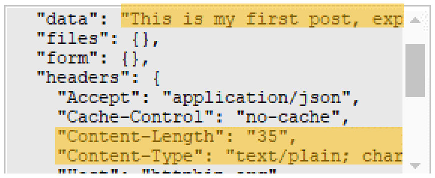
Request Body properties open.


Request Body | Request Body
Source Type | httpClient | SlotSource
Source | Slot Source
Data | This is my first post, expect more!
Clear Payload After | Neither

Step 4 Fill in the properties and click **Save**.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

The driver populates our test service (which echoes back the request content) from our **`Data`** slot, and automatically populates the **`Content-Length`** and **`Content-Type`**:

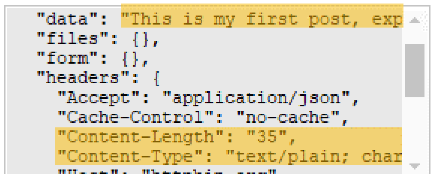

```
"data": "This is my first post, exp
"files": {},
"form": {},
"headers": {
  "Accept": "application/json",
  "Cache-Control": "no-cache",
  "Content-Length": "35",
  "Content-Type": "text/plain; char
```

Setting up an AutoHeader

An autoheader attempts to determine the `Content-Type`. Some additional automatic headers are available in the palette.

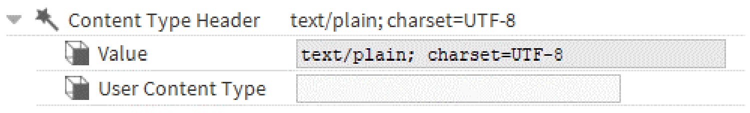
Prerequisites: The httpClient palette is open.

The driver populates our test service (which echoes back the request content) from our **`Data`** slot, and automatically populates the **`Content-Length`** and **`Content-Type`**:

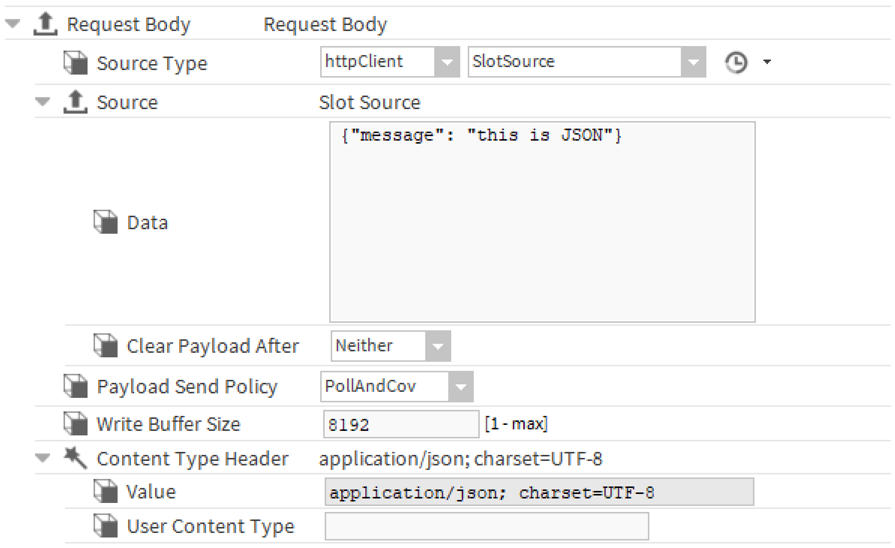

```
"data": "This is my first post, exp
"files": {},
"form": {},
"headers": {
  "Accept": "application/json",
  "Cache-Control": "no-cache",
  "Content-Length": "35",
  "Content-Type": "text/plain; char
```

Step 1 Notice that the `Content-Length` and `Content-Type` headers are automatically populated.

In another example, `Content-Type` defaults to text/plain. An AutoHeader component makes this possible. In this case, the `Content-Type` auto header is underneath the **`Request Body`**.

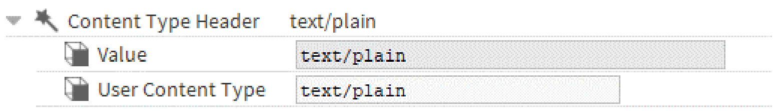

Content Type Header | text/plain; charset=UTF-8
Value | text/plain; charset=UTF-8
User Content Type |

This component attempts to determine the `Content-Type`. For example, if the **`Data`** slot is changed to: `{ "message": "this is JSON" }`, the auto header calculates the new `Content-Type` as `'application/json'`.



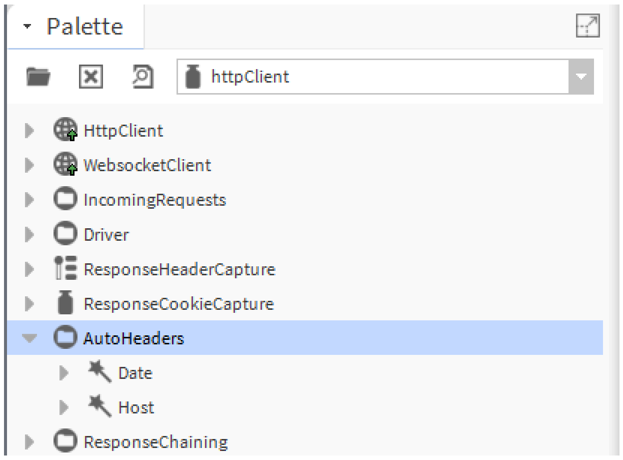
Request Body	Request Body
Source Type	httpClient / SlotSource
Source	Slot Source
Data	{"message": "this is JSON"}
Clear Payload After	Neither
Payload Send Policy	PollAndCov
Write Buffer Size	8192 [1 - max]
Content Type Header	application/json; charset=UTF-8
Value	application/json; charset=UTF-8
User Content Type	

Step 2 To override this behaviour, enter your own value into **`User Content Type`** slot

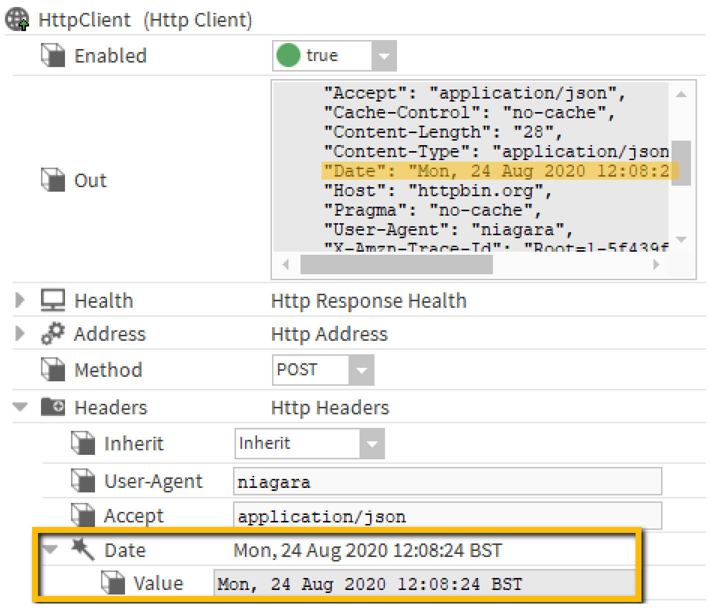


Content Type Header	text/plain
Value	text/plain
User Content Type	text/plain

Step 3 Locate the additional **AutoHeaders** in the palette.



Step 4 To apply each **AutoHeader**, drag the required component from the palette into the **Headers** folder:


HttpClient (Http Client)

Enabled	true
Out	
Health	Http Response Health
Address	Http Address
Method	POST
Headers	Http Headers
Inherit	Inherit
User-Agent	niagara
Accept	application/json
Date	Mon, 24 Aug 2020 12:08:24 BST
Value	Mon, 24 Aug 2020 12:08:24 BST

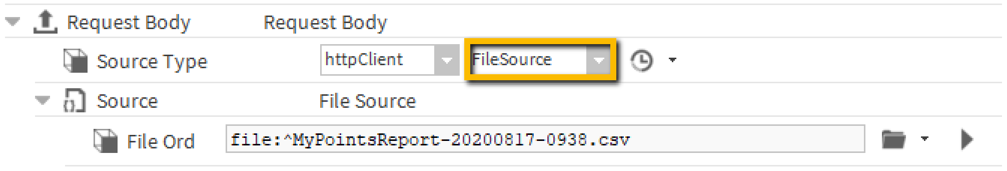
```
"Accept": "application/json",
"Cache-Control": "no-cache",
"Content-Length": "28",
"Content-Type": "application/json
"Date": "Mon, 24 Aug 2020 12:08:2
"Host": "httpbin.org",
"Pragma": "no-cache",
"User-Agent": "niagara",
"X-Amzn-Trace-Id": "Root=1-5f439f
```

Posting file content

This type of POST sends the contents of a file in the body of the POST request.

Prerequisites: You are connected to the station.

Step 1 Expand **HttpClient→Request Body→Source**.



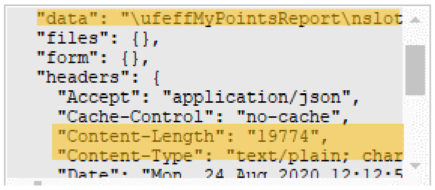
Request Body	Request Body
Source Type	httpClient FileSource
Source	File Source
File Ord	file:^MyPointsReport-20200817-0938.csv

Step 2 For **Source Type**, select `FileSource` from the drip-down list.

Step 3 Browse for or enter the **File Ord**.

Step 4 Right-click **HttpClient** and click **Actions→Send**.

The file contents become the body of the POST request.



```
"data": "\ufeffMyPointsReport\nslot
"files": {},
"form": {},
"headers": {
  "Accept": "application/json",
  "Cache-Control": "no-cache",
  "Content-Length": "19774",
  "Content-Type": "text/plain; char
  "Date": "Mon, 24 Aug 2020 12:12:5
```

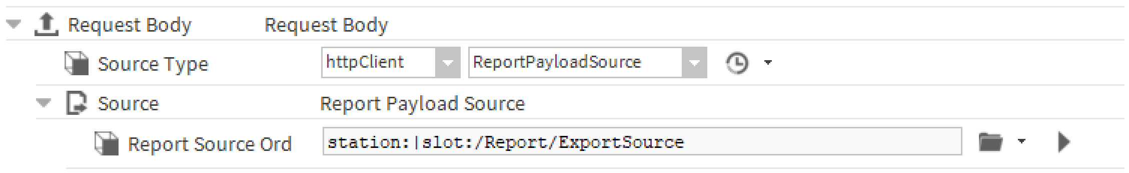
Once again, the `ContentType` auto header attempts to make a best guess from the first bytes of the file, and `Content-Length` is set.

Posting reports

This type of POST displays the contents of a report in the body of the POST request.

Prerequisites: You are connected to the station.

Step 1 Expand **HttpClient→Request Body→Source**.



Request Body	Request Body
Source Type	httpClient ReportPayloadSource
Source	Report Payload Source
Report Source Ord	station:|slot:/Report/ExportSource

Step 2 For **`Source Type`**, select `ReportPayloadSource` from the drip-down list.

Step 3 Browse for or enter the **`Report Source Ord`**.

Step 4 Right-click **HttpClient** and click **Actions→Send**.

The file contents become the body of the POST request.

Posting from data

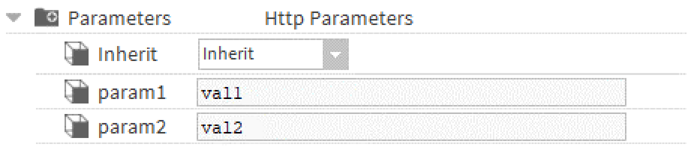
Some endpoint URL's, such as the targets for forms on webpages, expect the request body to contain url-encoded request parameters as the message body.

Prerequisites: You are connected to the station.

Step 1 Expand **HttpClient→Request Body→Source**.

Step 2 For **`Source Type`**, select `ParameterStringSource` from the drip-down list.

Step 3 Define your http request parameters, in the **Parameters** folder as usual:

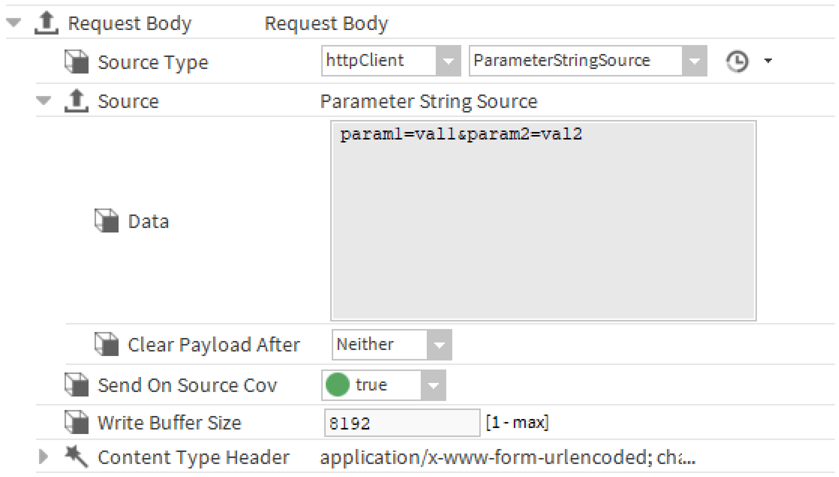


Parameters	Http Parameters
Inherit	Inherit
param1	val1
param2	val2

Step 4 Ensure that your client's **`Method`** is POST.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

The Data slot of the request body source is now read-only, and populated with the encoded parameter string:



Request Body	Request Body
Source Type	httpClient ParameterStringSource
Source	Parameter String Source
Data	param1=val1&param2=val2
Clear Payload After	Neither
Send On Source Cov	true
Write Buffer Size	8192 [1 - max]
Content Type Header	application/x-www-form-urlencoded; ch...

The driver automatically sets the `Content-Type` header to `application/x-www-form-urlencoded`.

Setting up a PUT request

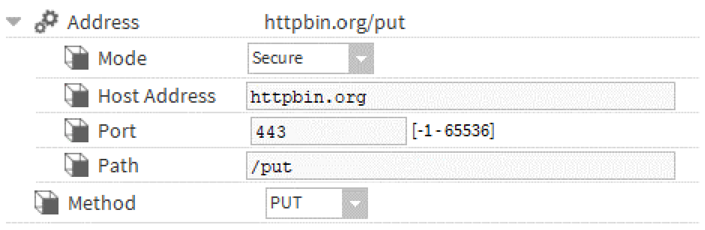
An Http PUT is identical to a POST in terms of the **HttpClient**. An API often makes a behavioural distinction between POST and PUT in terms of the creation and update of resources. If a PUT is required, there is no functional difference in this module, and all that is required is to change the client method to PUT.

Prerequisites: You are connected to the station, which has an HttpClient component.

Step 1 Expand **HttpClient** and change **`Method`** to `PUT`.

Step 2 Expand **`Address`**, enter the **`Host Address`** and **`Path`** and click **Save**.

The Address properties are configured for a PUT request.



Address	httpbin.org/put
Mode	Secure
Host Address	httpbin.org
Port	443 [-1 - 65536]
Path	/put
Method	PUT

Step 3 Expand **`Request Body`**.

Step 4 Fill in the properties and click **Save**.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

The driver populates our test service (which echoes back the request content) from our **`Data`** slot, and automatically populates the **`Content-Length`** and **`Content-Type`**:

Chaining client requests

Chaining client requests triggers events or secondary client requests after an initial **HttpClient** request has completed.

Prerequisites: The httpClient palette is open. An initial client request (GET, POST or PUT) has been configured.

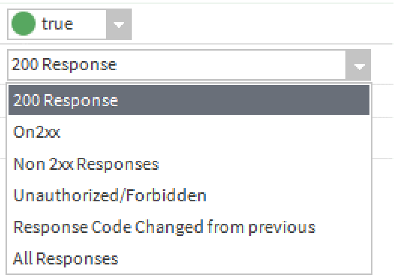
Step 1 Expand **ResponseChaining** and **Conditions** in the palette.

Step 2 Do one of the following:

- To add a request to an **HttpClient**, drag a **ResponseTrigger** to your **HttpClient** component in the station.
- To add a request to an **HttpClientDevice**, expand **HttpClientNetwork→HttpClientDevice→-Points→StringPoint** and add the **ResponseTrigger** to the **Proxy Ext** node in the Nav tree.

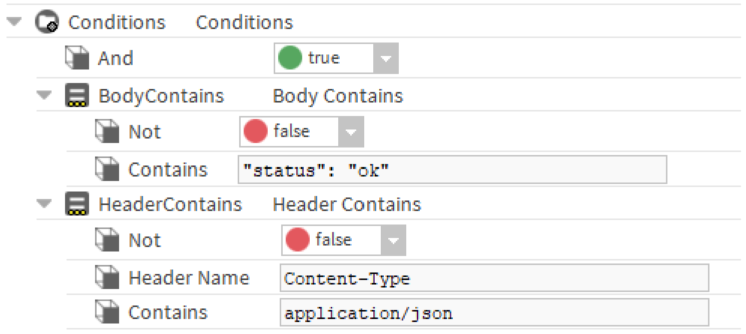
Step 3 To set up the logical criteria that need to be fulfilled, do one or more of the following:

- Select a response code from the **`Fire On`** drop-down list.



- Expand the **ResponseTrigger** and drag a condition (**BodyContains** and/or **HeaderContains**) from the palette to the **Conditions** folder.

Step 4 If you added a condition, expand it, set up the condition and configure **`Not`** appropriately.



Conditions	Conditions
And	true
BodyContains	Body Contains
Not	false
Contains	"status": "ok"
HeaderContains	Header Contains
Not	false
Header Name	Content-Type
Contains	application/json

In this example, the trigger only fires when the response received by the parent client:

- includes the text `status: ok` in the response body.
- includes a `Content-Type` header value of `application/json`.

All other responses prevent the trigger from firing.

Setting the **`Not`** value to `true` negates the defined logic of a condition. If you have multiple conditions defined, the default logic is to require all to be true. Set the **`And`** property to `false` and only one of the conditions needs to be true.

NOTE: Both the **`Fire On`** response code, and condition logic in the **Conditions** folder must both be true for the trigger to fire.

Configuring to fire secondary components

You can set up one or more secondary **HttpClient** components to send when the trigger logic fires. The **ResponseChain** is functionally the same as the **ResponseTrigger** component with some additional properties. It evaluates its logic each time a response is received by the parent.

Prerequisites: The httpClient palette is open.

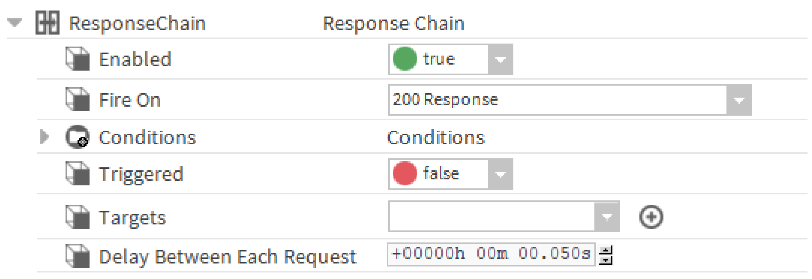
Step 1 Expand **ResponseChaining** in the palette.

Step 2 Do one of the following to configure secondary components:

- For an **HttpClient**, drag a **ResponseChain** from the palette to your **HttpClient** component in the station.
- For an **HttpClientDevice**, expand **HttpClientNetwork→HttpClientDevice→Points→StringPoint** and drag a **ResponseChain** from the palette to the **Proxy Ext** node in the Nav tree.

Step 3 Double-click the **ResponseChain**.

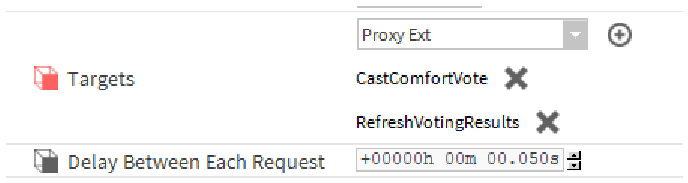
The component's **AX Property Sheet** opens.



ResponseChain	Response Chain
Enabled	true
Fire On	200 Response
Conditions	Conditions
Triggered	false
Targets	
Delay Between Each Request	+00000h 00m 00.050s

Step 4 To add a secondary client, expand the Targets drop-down list, select a client and click the plus button (⊕).

The secondary clients appear below the drop-down list.



Targets	Proxy Ext ⊕ CastComfortVote ✖ RefreshVotingResults ✖
Delay Between Each Request	+00000h 00m 00.050s

Step 5 To remove a secondary client, click the **X** next to the client.

Step 6 Configure the **`Delay Between Each Request`** and click **Save**.

This defines the minimum amount of time to elapse between the invocation of the **Send** action for each target client.

When a parent client receives a response, if the conditional logic and **`Fire On`** logic are met, each of the secondary clients in the **`Targets`** list sends in sequence.

Sending and receiving messages with the WebSocketClient

The **WebSocketClient** contains similar functionality to the standalone **HttpClient** component.

Prerequisites: The httpClient palette is open. **HttpClientService** has been added to station's **Service** container. Standalone client types in the **HttpClientService** have been enabled.

A regular conversation within the HTTP protocol consists of multiple requests and responses sent over separate underlying connections. A WebSocket is a persistent connection to an endpoint allowing full-duplex communications, where either the client or server side may send a message at any time.

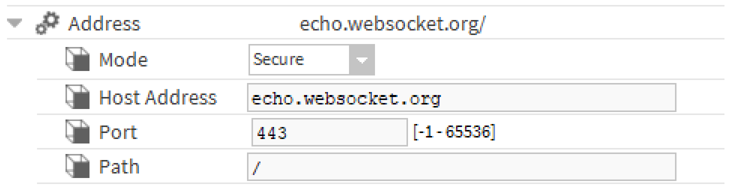
Step 1 Drag a **WebsocketClient** component from the palette to the station and double-click the component.

You may put it in the **Drivers** container.

The component's **AX Property Sheet** opens.

Step 2 Expand **Address**.

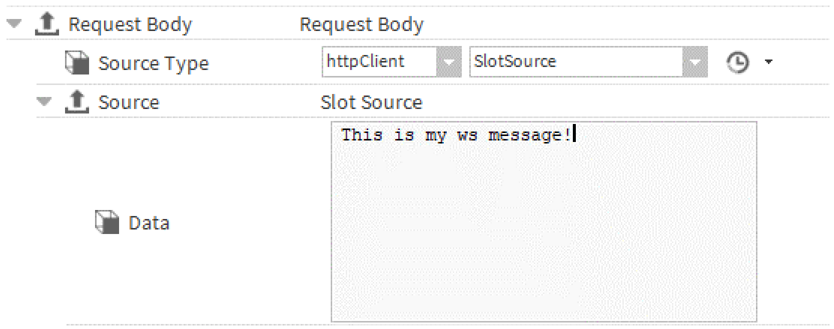
The **Address** properties open.



Address	echo.websocket.org/
Mode	Secure
Host Address	echo.websocket.org
Port	443 [-1 - 65536]
Path	/

Step 3 Expand **Request Body**.

The **Request Body** properties open.



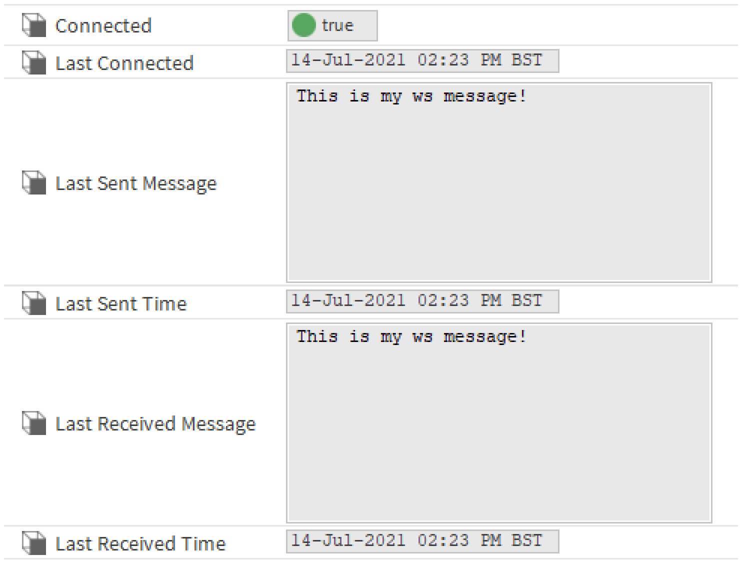
Request Body	Request Body
Source Type	httpClient SlotSource
Source	Slot Source
Data	This is my ws message!

Step 4 Populate the **`Data`** slot of the **Request Body**.

Notice that like the **HttpClient**'s other source types are available here.

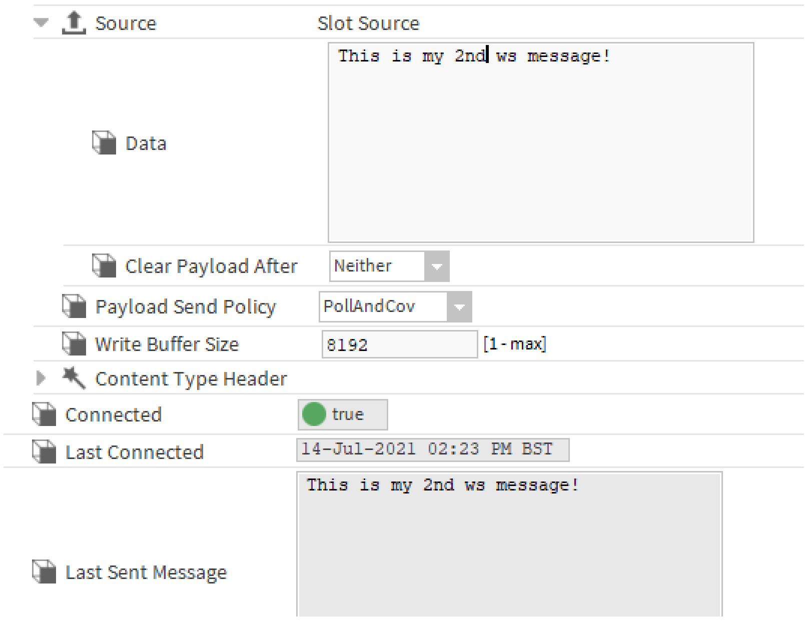
Step 5 Right-click the **WebsocketClient** and click **Actions→Send**.

The driver attempts to connect to the WebSocket and transmit the message. This example echos back all messages received.



Connected	true
Last Connected	14-Jul-2021 02:23 PM BST
Last Sent Message	This is my ws message!
Last Sent Time	14-Jul-2021 02:23 PM BST
Last Received Message	This is my ws message!
Last Received Time	14-Jul-2021 02:23 PM BST

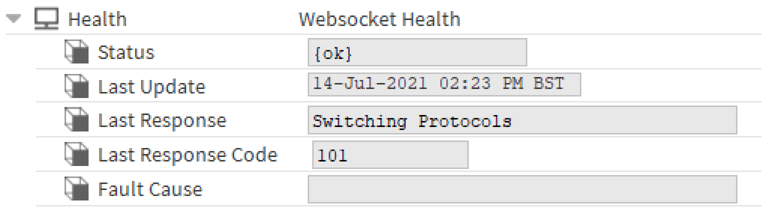
With the Payload Send Policy set to PollAndCov, any changes to the source message automatically result in a new message send:



Source	Slot Source
Data	This is my 2nd ws message!
Clear Payload After	Neither
Payload Send Policy	PollAndCov
Write Buffer Size	8192 [1 - max]
Content Type Header	
Connected	● true
Last Connected	14-Jul-2021 02:23 PM BST
Last Sent Message	This is my 2nd ws message!

Step 6 Expand **Health**.

The properties open.



Health	Websocket Health
Status	{ok}
Last Update	14-Jul-2021 02:23 PM BST
Last Response	Switching Protocols
Last Response Code	101
Fault Cause	

The **Health** component contains the same properties as the regular **HttpClient**, however the response code should only ever show the value of the initial upgrade request, which initiated the WebSocket connection.

The driver sends a regular keep-alive ping message while the connection is active.

Step 7 To disconnect from the WebSocket, right-click the component and click **Actions→Disconnect**.

Capturing incoming request messages

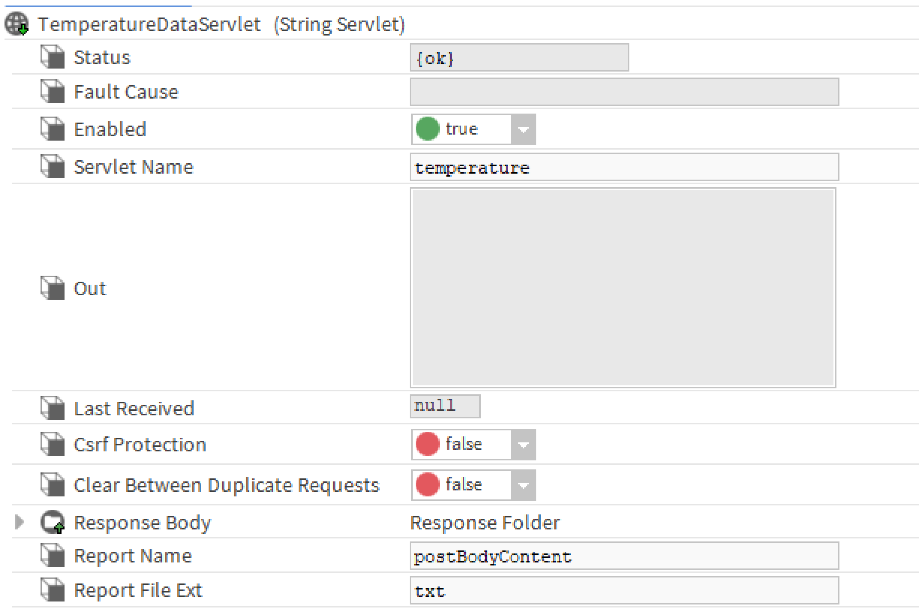
A **StringServlet** component captures incoming request messages.

Prerequisites: The httpClient palette is open.

Step 1 Expand **IncomingRequests** in the palette and add a **StringServlet** component to your station.

Step 2 Double-click the **StringServlet** component in the station.

The servlet's **AX Property Sheet** opens.



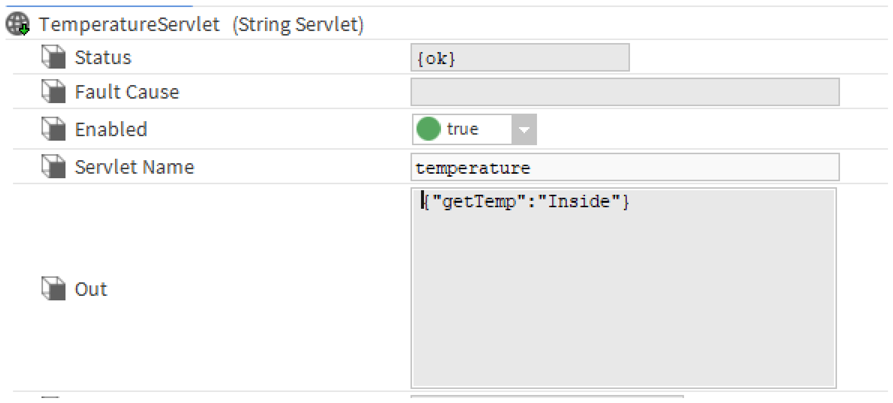
TemperatureDataServlet (String Servlet)	
Status	{ok}
Fault Cause	
Enabled	● true
Servlet Name	temperature
Out	
Last Received	null
Csrf Protection	● false
Clear Between Duplicate Requests	● false
Response Body	Response Folder
Report Name	postBodyContent
Report File Ext	txt

Step 3 Populate the **Servlet Name** with a name relevant to your requirements.

This name becomes the path of the http address to which clients send their requests.

Step 4 Right-click the servelet component and click **Actions→Send**.

The message body of any POST request appears in the `Out` property of the component.



TemperatureServlet (String Servlet)	
Status	{ok}
Fault Cause	
Enabled	● true
Servlet Name	temperature
Out	{"getTemp":"Inside"}

The command used for this example is: `curl -k -u username:password -X POST "https://127.0.0.1/temperature" -d '{"getTemp":"Inside"}'`

NOTE: The same user authentication used by all other station urls protects the address of the **StringServlet**. Additionally, the user account used to contact the **StringServlet** must have Operator Write permission on the **StringServlet** component.

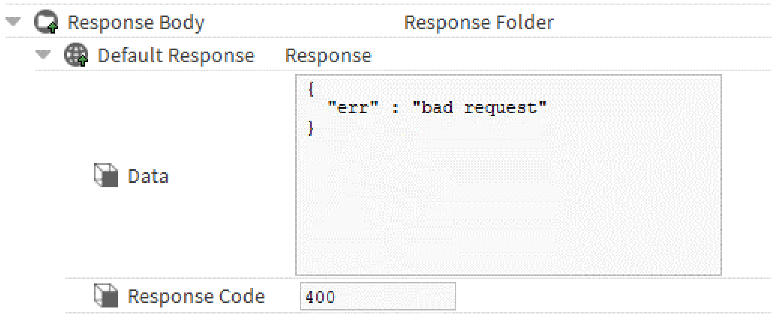
Defining the StringServlet response

The **ResponseBody** component of the **StringServlet** configures a response to be sent back to the remote client. This comes with a **DefaultResponse** component, which has slots for the body string (**`Data`**), and **`Response Code`**.

Prerequisites: A StringServlet exists in the station. The httpClient palette is open.

Step 1 Double-click the **StringServlet** and expand **Response Body**.

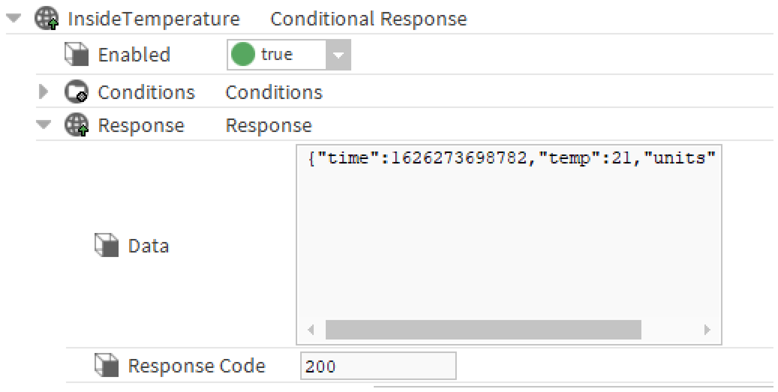
The **Response Body** properties open.



Response Body		Response Folder
Default Response	Response	
Data	{ "err" : "bad request" }	
Response Code	400	

Step 2 To add one or more alternative responses, drag a **ConditionalResponse** component from the palette to the **Response Body** folder and double-click the **ConditionalResponse** component.

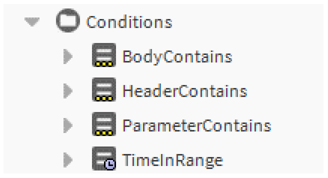
The response properties open.



InsideTemperature	Conditional Response
Enabled	true
Conditions	Conditions
Response	Response
Data	{"time":1626273698782,"temp":21,"units"
Response Code	200

Step 3 Enter an alternative **Data** property and **Response Code**.

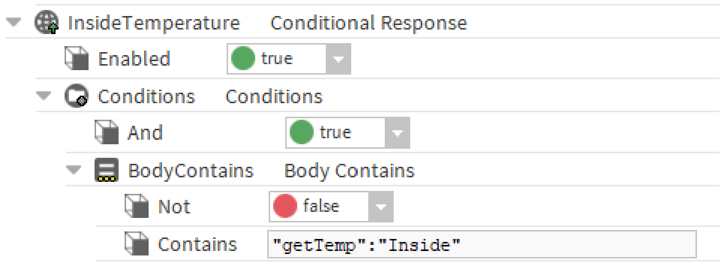
This example links the output of a JSONSchema component to the **Data** slot. For this **ConditionalResponse** to be used, some conditions must be defined. Several example conditions are available in the palette:



- Conditions
  - BodyContains
  - HeaderContains
  - ParameterContains
  - TimeInRange

Step 4 Expand **IncomingRequests→ConditionalResponse→Conditions** and drag a condition, such as **BodyContains** from the palette to the Conditions folder under **ConditionalResponse** and double-click the condition.

The condition's properties open.



InsideTemperature	Conditional Response
Enabled	true
Conditions	Conditions
And	true
BodyContains	Body Contains
Not	false
Contains	"getTemp":"Inside"

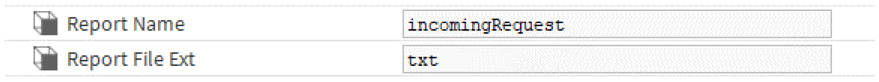
Step 5 Set the **`Not`** value to `true`.

This negates the defined logic.

If you have multiple conditions defined, the default logic requires that all must be `true`. Set the **`And`** slot to `false` and only one of the conditions needs to be true.

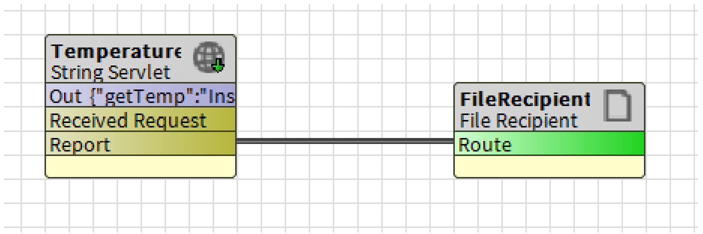
Step 6 To configure a response report, enter a filename for **`Report Name`** and define the file extension using the **`Report File Ext`** property.

These are the last properties at the bottom of the **StringServlet AX Property Sheet**.



Report Name	incomingRequest
Report File Ext	txt

Step 7 To create a file to capture each request in the station home folder, link this topic to an appropriate recipient, such as a report FileRecipient.



Step 8 Right-click the servlet component and click **Actions→Send**.

In the example, when we repeat the same external request, the conditional response returns:

```
curl -k -u MrBasic:Manager123 -X POST "https://127.0.0.1/temperature" -d
'{"getTemp":"Inside"}'
```

```
{"time":1626273698782,"temp":21,"units":"°C"}
```

It is also possible to send GET requests to a **StringServlet**. The functionality is the same, except no **Request Body** can be posted.

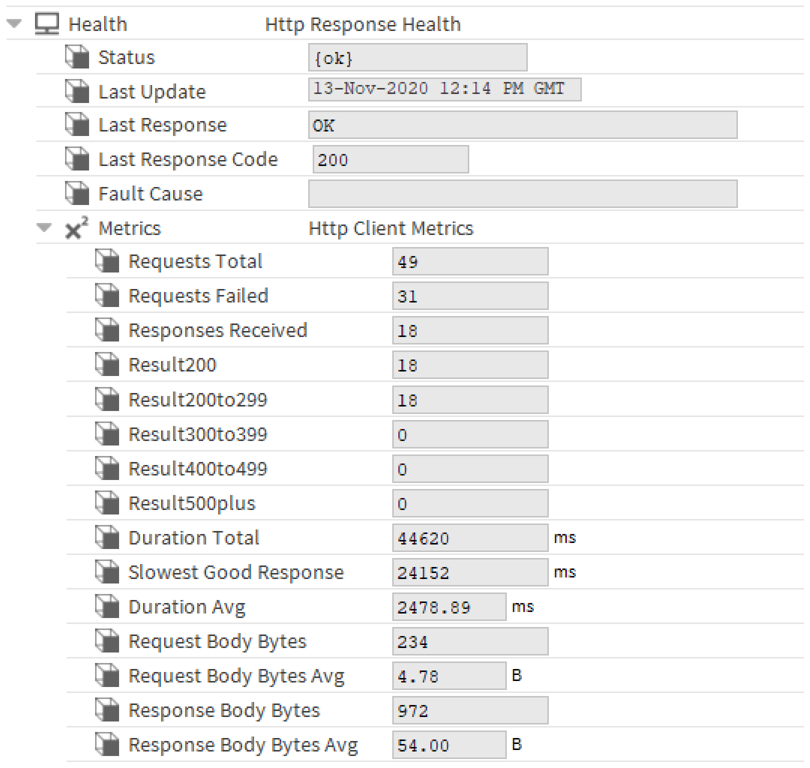
Monitoring request and response metrics

The **Health** component of a client and proxy extension contains a **Metrics** component with several properties to use for analysis and fault diagnosis. These include request success versus failure counts, duration and size of requests and responses, and a breakdown of responses by code.

Prerequisites: Your station includes request and response components

Step 1 Expand the **Health→Metrics** component under the **HttpClient AX Property Sheet**.

The **Health** and **Metrics** properties open.



Health	Http Response Health	
Status	{ok}	
Last Update	13-Nov-2020 12:14 PM GMT	
Last Response	OK	
Last Response Code	200	
Fault Cause		
Metrics	Http Client Metrics	
Requests Total	49	
Requests Failed	31	
Responses Received	18	
Result200	18	
Result200to299	18	
Result300to399	0	
Result400to499	0	
Result500plus	0	
Duration Total	44620	ms
Slowest Good Response	24152	ms
Duration Avg	2478.89	ms
Request Body Bytes	234	
Request Body Bytes Avg	4.78	B
Response Body Bytes	972	
Response Body Bytes Avg	54.00	B

Step 2 Review the statistics.

Step 3 To clear these values, right-click **Health** and click **Actions→Reset**.

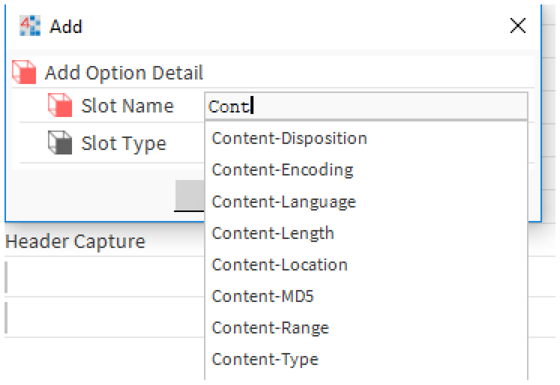
Capturing Response Headers

You may have the need to capture headers from a response. This allows linking within **Wire Sheet** logic, perhaps to use as a header value on another client request.

Prerequisites: Your station as an **HttpClient**.

Step 1 Expand **HttpClient**, right-click **Headers** and click **Actions→Add**

The **Add** window opens.



Slot Name	Cont
Slot Type	

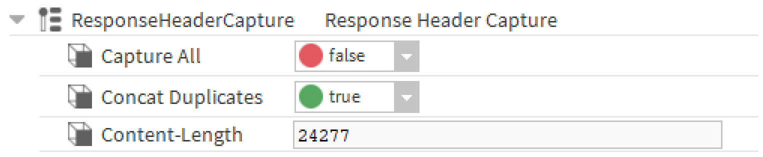
Content-Disposition
Content-Encoding
Content-Language
Content-Length
Content-Location
Content-MD5
Content-Range
Content-Type

Step 2 Start typing the name of the header in **Slot Name**.

Auto-complete may help here.

Step 3 Right-click **HttpClient** and click **Actions→Send**.

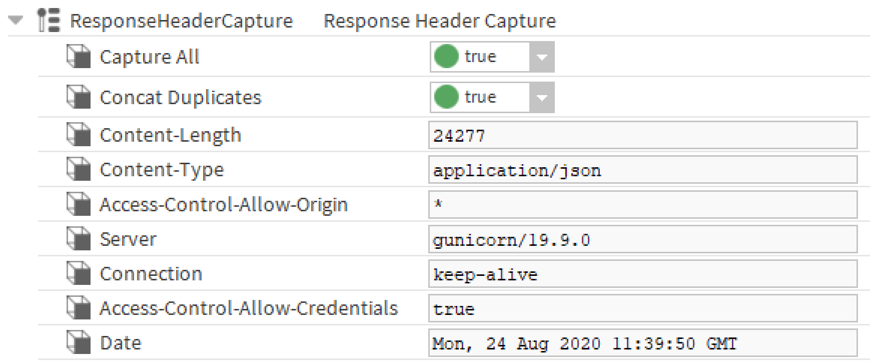
The driver captures the value from the response.



ResponseHeaderCapture	Response Header Capture
Capture All	false
Concat Duplicates	true
Content-Length	24277

Step 4 Switch **`Capture All`** to `true` and send again.

The driver creates and updates slots for all received headers.



ResponseHeaderCapture	Response Header Capture
Capture All	true
Concat Duplicates	true
Content-Length	24277
Content-Type	application/json
Access-Control-Allow-Origin	*
Server	gunicorn/19.9.0
Connection	keep-alive
Access-Control-Allow-Credentials	true
Date	Mon, 24 Aug 2020 11:39:50 GMT

If **`Concat Duplicates`** is `true`, and a response contains two headers with the same name, the driver concatenates the values as a csv string.

Capturing cookies

You may need to capture cookie values from a response. This allows linking within **Wire Sheet** logic, perhaps to use as a cookie value on another client request.

Prerequisites: The **`httpClient`** palette is open.

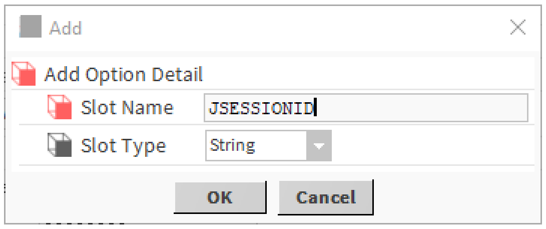
Step 1 Drag a **ResponseCookieCapture** component from the palette to the **HttpClient** or **Http Proxy Client Ext**.

For example, you may receive these http headers in a response:

```
Set-Cookie: JSESSIONID=456789; Expires=Wed, 09 Jun 2021 10:18:14 GMT
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
```

Step 2 Expand **HttpClient**, right-click **Headers** and click **Actions→Add**.

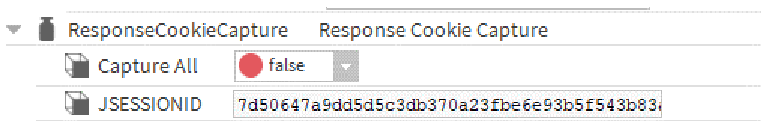
The **Add** window opens.



Add Option Detail	
Slot Name	JSESSIONID
Slot Type	String

Step 3 Right-click **HttpClient** and click **Actions→Send**.

The driver extracts the response cookies.



ResponseCookieCapture Response Cookie Capture

Capture All false

JSESSIONID 7d50647a9dd5d5c3db370a23fbe6e93b5f543b83

Step 4 Switch **Capture All** to `true` and send again.

For example:

```
JSESSIONID: 456789
SID: 31d4d96e407aad42
```

The driver creates and updates slots for all received cookies and discards all other cookies attributes beyond the value.

Troubleshooting

Several features are available for troubleshooting.

DebugService

To diagnose problems with HTTP client requests, you may set the following categories in the **DebugService** to the FINE level for logging, and inspect the output in the **Application Director**:

- httpClient
- httpClient.license
- httpClient.messageQueue
- httpClient.transport
- httpClient.ws

Certificate management

At times an HTTPS connection may fail due to an untrusted certificate issued by the remote server. You may review and approve these exceptions under **Services→PlatformServices→CertManagerService**.

Chapter 4 Security

Topics covered in this chapter

Security involves user and server authentication as well as data encryption.

User authentication

Many APIs and web services protect their functionality and data by requiring various means of authentication.

HTTP client provides these authentication methods:

- HTTP Basic
- HTTP Digest
- Niagara SCRAM-SHA
- Bearer token
- Cookies from a previous request

Using HTTP Basic authentication

HTTP Basic Authentication is the least secure form of authentication supplied in the client.

Prerequisites: You are working in Workbench and are connected to the station with an **HttpClientNetwork**.

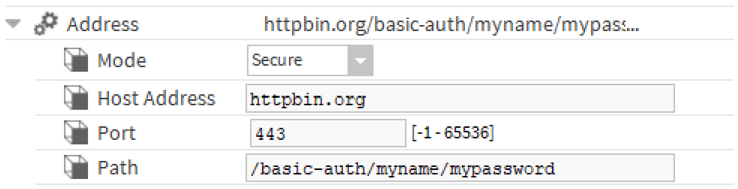
The username and password are included in the requests Authorization header in the form:

`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

where the username:password are Base64 encoded.

Step 1 Double-click **HttpClient** and expand **Address**.

The **Property Sheet** opens.

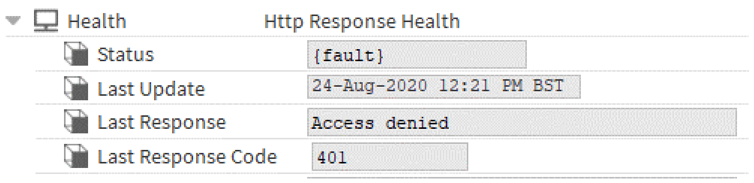


Address	httpbin.org/basic-auth/myname/mypas:...
Mode	Secure
Host Address	httpbin.org
Port	443 [-1 - 65536]
Path	/basic-auth/myname/mypassword

In this example, the url has been changed to one that is protected by HTTP Basic authorization.

Step 2 Set **Mode** to `Secure`.

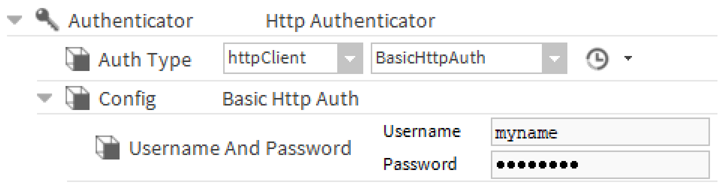
This setting is required because credentials are not encrypted and the encoding is simple to reverse engineer. If the client's **Mode** is set to `Insecure`, the HTTP client will fail to send with this error message: `Exception occurred, Failed to build request BasicHttpAuth requires HTTPS` and **HttpClient Health** displays:



Health	Http Response Health
Status	{fault}
Last Update	24-Aug-2020 12:21 PM BST
Last Response	Access denied
Last Response Code	401

Step 3 Expand **Config→Drivers→HttpClientNetwork**, double-click the **HttpClientDevice** and expand **Authenticator**.

The properties expand.



Authenticator	Http Authenticator	
Auth Type	httpClient	BasicHttpAuth
Config	Basic Http Auth	
Username And Password	Username	myname
	Password	••••••••

Step 4 Select `BasicHttpAuth` from the **`Auth Type`** drop-down list and click **Save**

The driver updates the **`Config`** options.

Step 5 Expand **Config**, set up **`Username And Password`** credentials and click **Save**.

Step 6 Right-click **HttpClient** and click **Actions→Send**.

Using Bearer Token authentication

Bearer token authentication is the method often used when an API requires a token string to identify the user or user session. This procedure uses Bearer Token authentication.

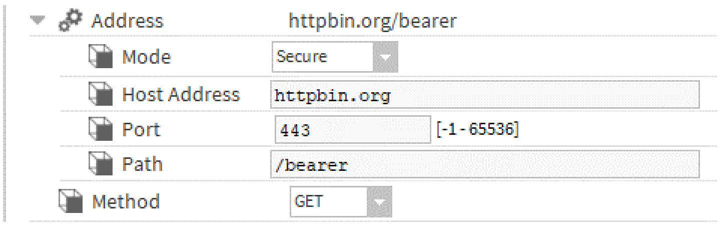
Prerequisites: You are working in Workbench and are connected to the station with an **HttpClientNetwork**. You have the token to authorize Bearer Token authentication.

This authentication method is included in the Authorization header as follows:

`Authorization: Bearer xxx`

Step 1 Double-click **HttpClient** and expand **Address**.

The **Property Sheet** opens.



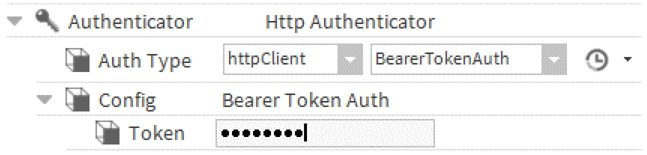
Address	httpbin.org/bearer	
Mode	Secure	
Host Address	httpbin.org	
Port	443	[-1 - 65536]
Path	/bearer	
Method	GET	

In this example, the address has been changed to a url protected by bearer token auth.

Step 2 Set **`Mode`** to `Secure` and **`Path`** to `/bearer`.

Step 3 Expand **Config→Drivers→HttpClientNetwork**, double-click the **HttpClientDevice** and expand **Authenticator**.

The properties expand.



Authenticator	Http Authenticator	
Auth Type	httpClient	BearerTokenAuth
Config	Bearer Token Auth	
Token	••••••••	

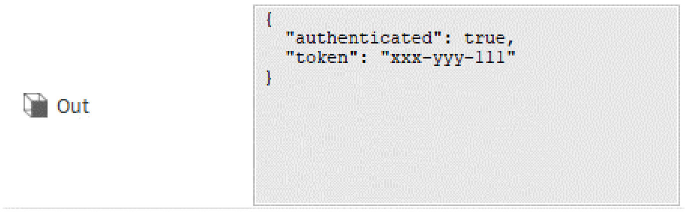
Step 4 Select `BearerTokenAuth` from the **`Auth Type`** drop-down list and click **Save**

The driver updates the **`Config`** options.

Step 5 Expand **Config**, enter the **`Token`** and click **Save**.

Step 6 Right-click **HttpClient** and click **Actions→Send**.

The driver sends the request and the Out slot reports success.


Out

```
{
  "authenticated": true,
  "token": "xxx-yyy-111"
}
```

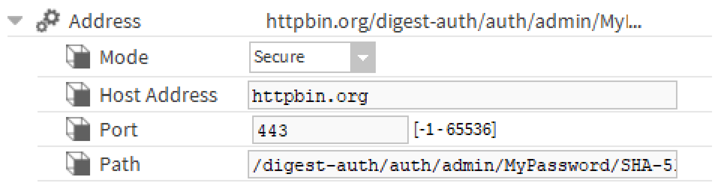
Using Digest authentication

Digest authentication involves a hash function applied to the user credentials.

Prerequisites: You are working in .Workbench and are connected to the station with an **HttpClientNetwork**.

Step 1 Double-click **HttpClient** and expand **Address**.

The **Property Sheet** opens.



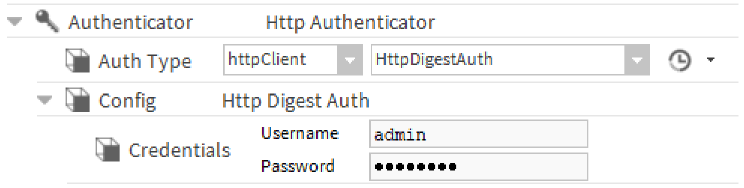
Address	httpbin.org/digest-auth/auth/admin/MyI...
Mode	Secure
Host Address	httpbin.org
Port	443 [-1 - 65536]
Path	/digest-auth/auth/admin/MyPassword/SHA-5

In this example, the address has been changed to a url protected by digest auth.

Step 2 Set **`Mode`** to `Secure` and **`Path`** appropriately.

Step 3 Expand **Config→Drivers→HttpClientNetwork**, double-click the **HttpClientDevice** and expand **Authenticator**.

The properties expand.



Authenticator	Http Authenticator
Auth Type	httpClient / HttpDigestAuth
Config	Http Digest Auth
Credentials	Username: admin; Password: ••••••••

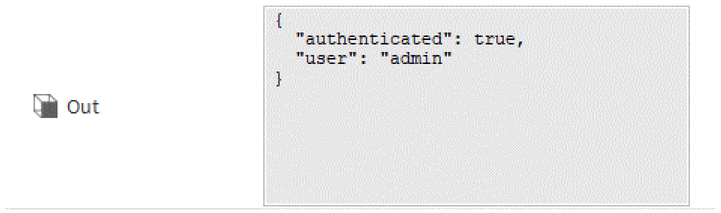
Step 4 Select `HttpDigestAuth` from the **`Auth Type`** drop-down list and click **Save**

The driver updates the **`Config`** options.

Step 5 Expand **Config** and set up **`Credentials`** (**`Username`** and **`Password`**) and click **Save**.

Step 6 Right-click **HttpClient** and click **Actions→Send**.

The driver sends the request and the **`Out`** slot reports success.



Out

```
{
  "authenticated": true,
  "user": "admin"
}
```

NOTE: auth-int digest authentication is not currently supported.

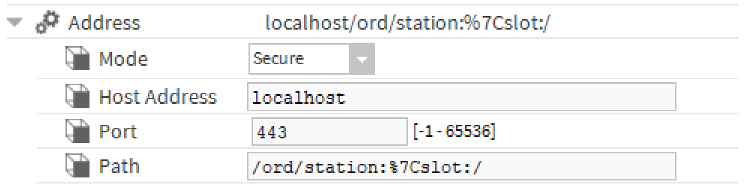
Using Niagara SCRAM-SHA authentication

The default authenticator on a Niagara users credentials is SCRAM-SHA Digest, which is a more complex variant of Digest authentication.

Prerequisites: You are working in Workbench and are connected to the station with an **HttpClientNetwork**.

Step 1 Double-click **HttpClient** and expand **Address**.

The **Property Sheet** opens.



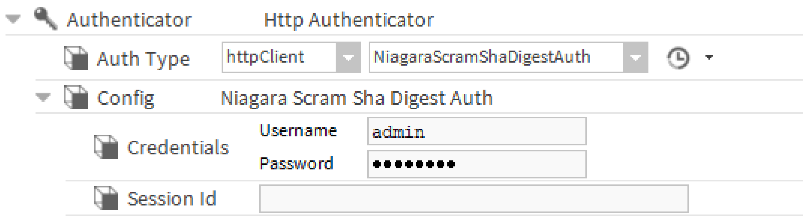
Address	localhost/ord/station:%7Cslot:/
Mode	Secure
Host Address	localhost
Port	443 [-1 - 65536]
Path	/ord/station:%7Cslot:/

In this example, the address has been changed to a url protected by digest auth.

Step 2 Set **Mode** to `Secure` and **Path**, for example, to `/ord/station:%7Cslot:/`.

Step 3 Expand **Config→Drivers→HttpClientNetwork**, double-click the **HttpClientDevice** and expand **Authenticator**.

The properties expand.



Authenticator	Http Authenticator	
Auth Type	httpClient	NiagaraScramShaDigestAuth
Config	Niagara Scram Sha Digest Auth	
Credentials	Username	admin
	Password	••••••••
Session Id		

Step 4 Select `HttpScramShaDigestAuth` from the **Auth Type** drop-down list and click **Save**

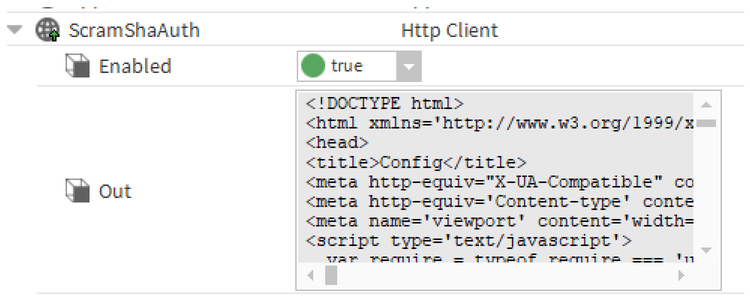
The driver updates the **`Config`** options.

Step 5 Expand **Config** and set up **`Credentials`** (**`Username`** and **`Password`**) and click **Save**.

We do not recommend the use of admin accounts for this utility.

Step 6 Right-click **HttpClient** and click **Actions→Send**.

The driver sends the request and the **`Out`** slot reports success.


ScramShaAuth Http Client
Enabled true
Out
```
<!DOCTYPE html>
<html xmlns='http://www.w3.org/1999/x
<head>
<title>Config</title>
<meta http-equiv="X-UA-Compatible" co
<meta http-equiv='Content-type' conte
<meta name='viewport' content='width=
<script type='text/javascript'>
  var require = typeof require === 'u
```

The read-only **hasSession** property populates on a successful connection.

It automatically becomes invalid if the session becomes inactive or it expires. In this instance, the client receives a 401 error and automatically repeats the SCRAM-SHA handshake on the next request attempt.

Step 7 To manually clear the session, right-click**Authenticator→Config** and click **Actions→Clear Session**.

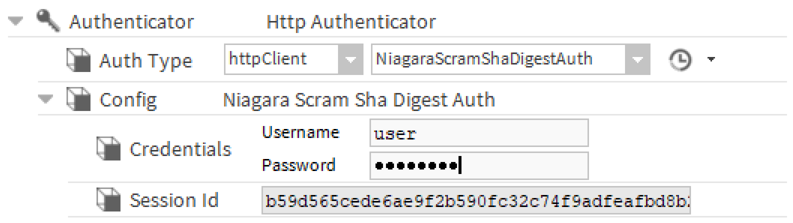
Using the Response Cookie authenticator

Many websites make use of an initial authentication method to create a user session, and then make use of session cookies to authenticate the user for subsequent requests.

Prerequisites: You are working in .Workbench and are connected to the station with an **HttpClientNetwork**. The **httpClient** palette is open.

Step 1 Set up an initial request that authenticates the user.

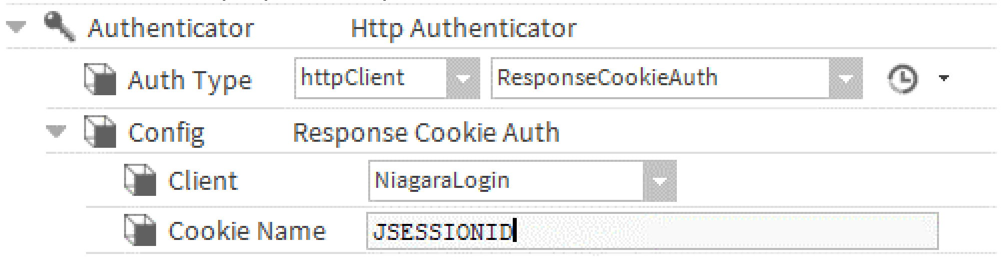
For this example, set up a station with SCRAM-SHA authentication to a station url.


Authenticator Http Authenticator
Auth Type httpClient NiagaraScramShaDigestAuth
Config Niagara Scram Sha Digest Auth
Credentials Username user
Password ••••••••
Session Id b59d565cede6ae9f2b590fc32c74f9adfeafbd8b

This creates a session.

Step 2 Drag a second **HttpClient** from the palette to the station and expand its **Authenticator** slot.

The **Authenticator** properties expand.


Authenticator Http Authenticator
Auth Type httpClient ResponseCookieAuth
Config Response Cookie Auth
Client NiagaraLogin
Cookie Name JSESSIONID

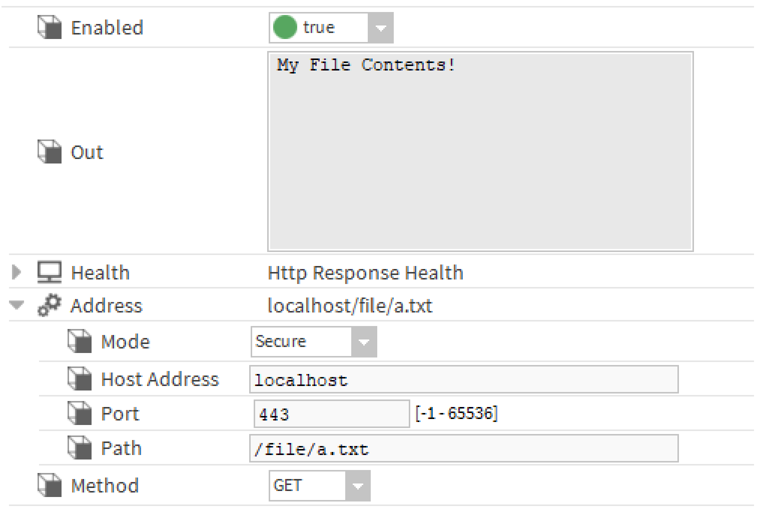
Step 3 Change **Auth Type** to ResponseCookieAuth and click **Save**.

The driver updates the **Config** options.

Step 4 Pick the **Client** to use to create the session, define the **Cookie Name** and click **Save**.

Step 5 Right-click **HttpClient** and click **Actions→Send**.

The second client is able to access a protected url using the session cookie.



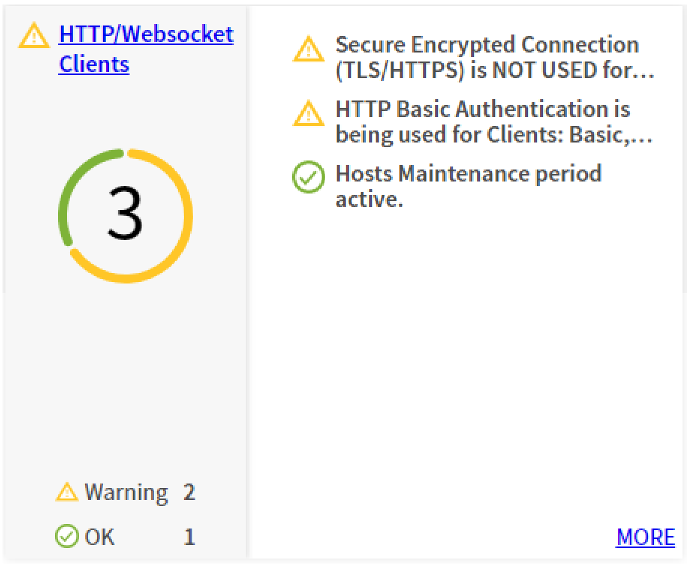
Enabled	true
Out	My File Contents!
Health	Http Response Health
Address	localhost/file/a.txt
Mode	Secure
Host Address	localhost
Port	443 [-1 - 65536]
Path	/file/a.txt
Method	GET

NOTE: The **Response Trigger** and **Response Chain** components are useful if you need the first request to specifically trigger the **Send** action on a second.

Security dashboard

The Niagara **SecurityService** dashboard presents warnings for HTTP and WebSocket clients within the station.

Figure 3 Example of SecurityService messages



HTTP/Websocket Clients

3

Secure Encrypted Connection (TLS/HTTPS) is NOT USED for...

HTTP Basic Authentication is being used for Clients: Basic,...

Hosts Maintenance period active.

Warning 2

OK 1

MORE

- `Secure Encrypted Connection (TLS/HTTPS) is NOT USED...` Please use https or wss where possible.
- `HTTP Basic Authentication is being used for Clients:...` Please use an authentication method other than Basic Authentication
- `Hosts Maintenance period expired or near expiry`. Keep your License Maintenance agreement up-to-date.
- `Non driver client types are enabled`. Keeping all clients within the driver container makes it easier to manage the user access to http clients.
- `Compatible TLS enabled in okhttp transport`. A more secure TLS scheme is favoured.

Chapter 5 Components

Topics covered in this chapter

Components include services, folders and other model building blocks associated with a module. You drag them to a property or wire sheet from a palette. Views are plugins that can be accessed by double-clicking a component in the Nav tree or right-clicking a component and selecting its view from the **Views** menu.

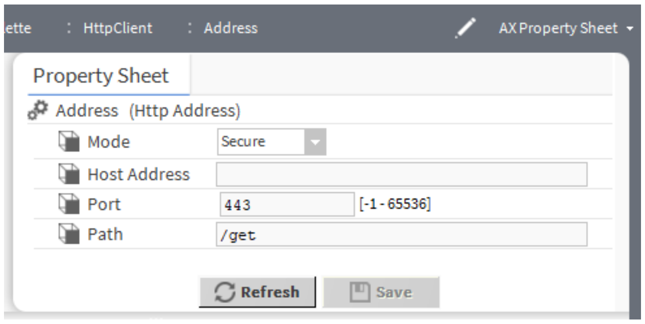
The component and view topics that follow appear as context-sensitive help topics when accessed by:

- Right-clicking on the object and selecting **Views→Guide Help**
- Clicking **Help→Guide On Target**

Address (httpClient-HttpAddress)

this component can configure each request to have a different HTTP address.

Figure 4 Address properties



Address (Http Address)
Mode Secure
Host Address
Port 443 [-1 - 65536]
Path /get

To access these properties, double-click **HttpClient** and click **Address**.

Property	Value	Description
Mode	drop-down list	Selects the security mode. `Secure`: Secure mode refers to https on port 443 by default. `Insecure`: Insecure mode means http without SSL and assumes port 80 by default.
Host Address	url	Defines the client's url address and parameters.
Port	number (defaults to 443)	Defines the communication port.
Path	text	Defines the path to the resource in the web service (that is, the path after the host address).

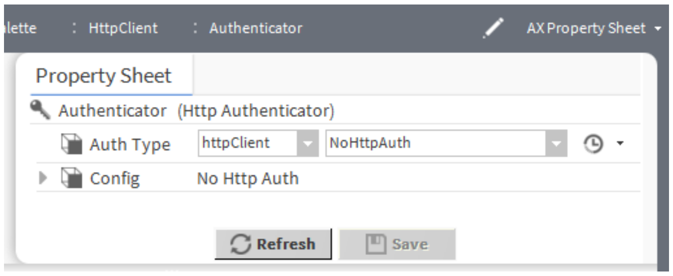
Action

Populate From Url automatically populates the host address.

Authenticator (httpClient-Http Authenticator)

This component configures the authenticator.

Figure 5 Authenticator properties



Authenticator (Http Authenticator)
Auth Type httpClient NoHttpAuth
Config No Http Auth

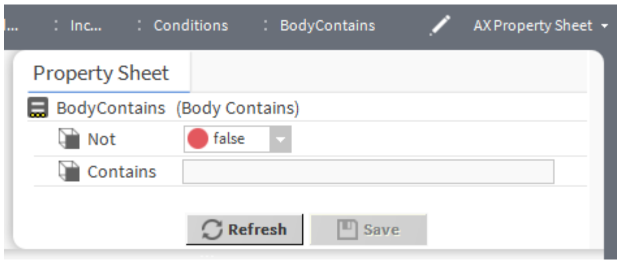
To access these properties, double-click **HttpClient** and click **Authenticator**.

Property	Value	Description
Auth Type	drop-down lists (default to `httpClient` and `NoHttpAuth`)	Configures the authentication method: `BasicHttpAuth` is the least secure form of authentication supplied in the client. As credentials are not encrypted and the encoding is simple to reverse engineer, the HTTP client fails to send if the client's **Mode** is set to insecure. `BearerTokenAuth` is the method used when an API requires a token string to identify the user or user session. `HttpDigestAuth` involves a 'hash function' applied to the user's credentials. `NiagaraScramShaDigestAuth` is a more complex variant of Digest authentication. It serves as the default authenticator for a Niagara user's credentials. `NoHttpAuth` configures no HTTP authentication. `ResponseCookieAuth`: Many websites make use of an initial authentication method to create a user session, and then make use of session cookies to authenticate the user for subsequent requests. To make use of this technique with an HTTP client, first set up an initial request, which authenticates the user.
Config	additional properties	Contains additional configuration items. To switch authentication methods, select from the various types in the **Auth Type** slot and save. The **Config** slot updates allowing further settings to be applied. This slot is its own component. Refer to "Config (httpClient-NoHttpAuth)".

Body Contains (httpClient-BodyContains)

This component defines the conditional response.

Figure 6 Body Contains properties



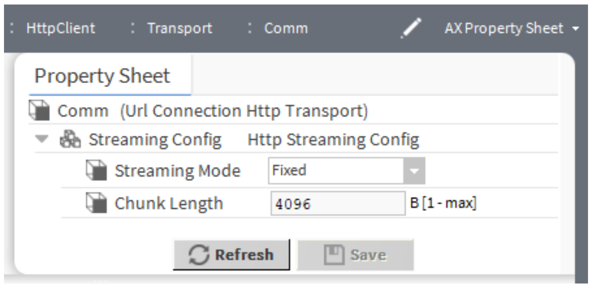
BodyContains (Body Contains)
Not: false
Contains:

Property	Value	Description
Not	`true` or `false` (default)	Indicates if the condition will be used (`false`) for the response or not (`true`).
Contains	text	Defines a string to search for in the request body.

Comm (httpClient-UrlConnectionHttpTransport)

This component configures the data streaming mode.

Figure 7 Comm properties



Comm (Url Connection Http Transport)
Streaming Config Http Streaming Config
Streaming Mode Fixed
Chunk Length 4096 B [1 - max]

To access these properties, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice→Transport** and double-click **Comm**.

Property	Value	Description

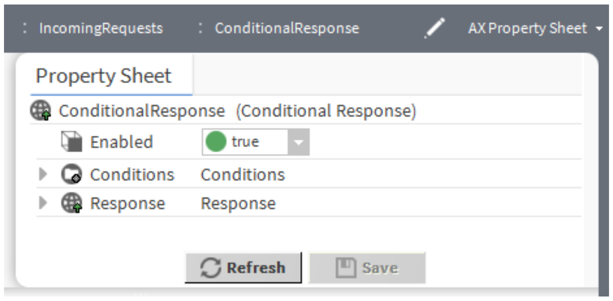
Streaming Mode	drop-down list (defaults to `Fixed`)	Selects how to stream the data.
		`Fixed`
		`Chunked`
		`Disable Monitor`

Chunk Length	number (defaults to 4096)	Configures the length of the record.

Conditional Response (httpClient-ConditionalResponse)

This component defines conditions that govern responses. Using conditions, you can configure one or more alternative responses. Several example conditions are available in the **Conditions** folder in the palette.

Figure 8 Conditional Response properties



ConditionalResponse (Conditional Response)
Enabled true
Conditions Conditions
Response Response

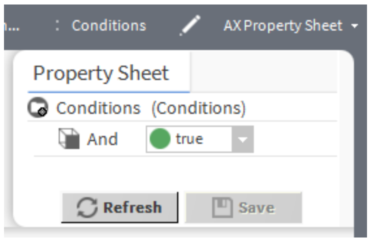
In addition to the standard properties (Enabled), this component provides these properties.

Property	Value	Description
Conditions	additional properties	Provides a second way to define a trigger criterion by adding one or two conditions from the palette (**BodyContains** and **HeaderContains**), then configures the `And` Boolean property appropriately. For property descriptions refer to "Conditions (httpClient-HttpConditions)".
Response	additional properties	Configures the response to be sent back to the remote client. For property descriptions, refer to "Default Response (httpClient-Response)".

Conditions (httpClient-Conditions)

This component configures the use of a **ConditionalResponse**.

Figure 9 Conditions property



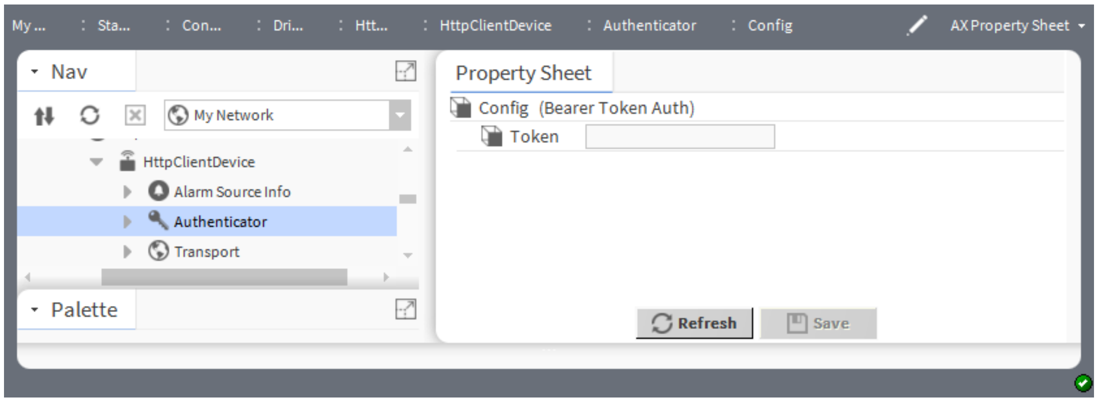
Conditions (Conditions)
And true

Property	Value	Description
And	`true` (default) or `false`	Configures how the software treats multiple conditions. `true`: ands conditions. `false`: ors conditions, only one condition needs to pass.

Config (httpClient-Bearer Token Auth)

This component defines an authorization token.

Figure 10 Bearer Token Auth property



Config (Bearer Token Auth)
Token

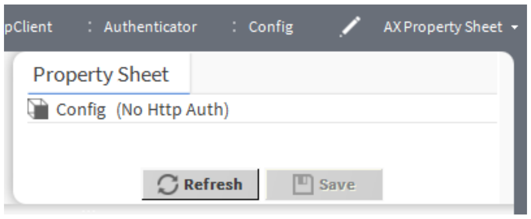
To access this property, expand **Config→Drivers→HttpClientNetwork→Authenticator** and click **Config**.

Property	Value	Description
Token	text	Defines the configuration token.

Config (httpClient-NoHttpAuth)

This component is a sup-component of the Authenticator.

Figure 11 No Http Auth property



Config (No Http Auth)

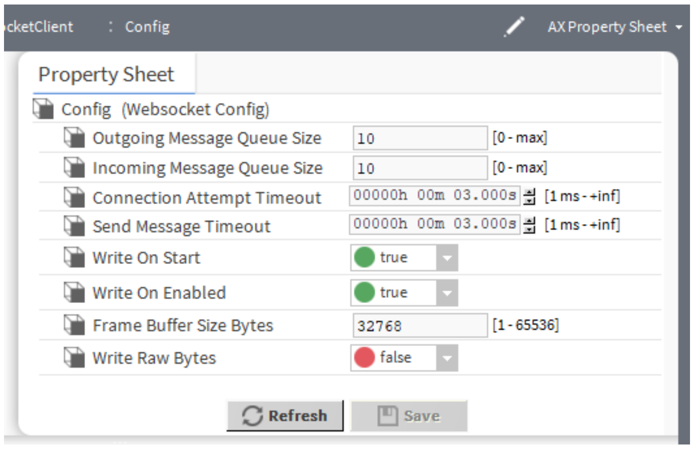
To access this component, double-click **HttpClient→Authenticator** and double-click .**Config**.

Property	Value	Description
Config		No properties to configure.

Config (httpClient-WebsocketConfig)

This component contains configures the web socket.

Figure 12 Config properties



Config (Websocket Config)

Outgoing Message Queue Size	10	[0 - max]
Incoming Message Queue Size	10	[0 - max]
Connection Attempt Timeout	00000h 00m 03.000s	[1 ms - +inf]
Send Message Timeout	00000h 00m 03.000s	[1 ms - +inf]
Write On Start	true	
Write On Enabled	true	
Frame Buffer Size Bytes	32768	[1 - 65536]
Write Raw Bytes	false	

To access these properties, expand **WebsocketClient** and double-click **Config**.

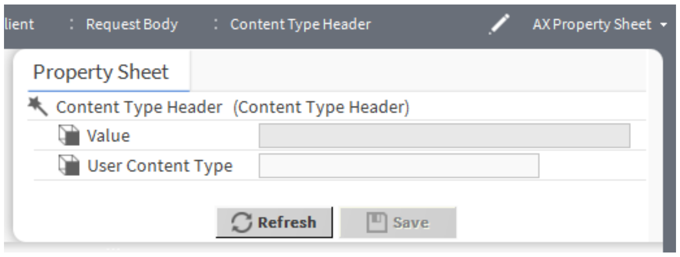
Property	Value	Description
Outgoing Message Queue Size	number (Range 0–max, Defaults 10)	Configures a maximum queue size for outgoing messages. This is required to cope with rapid changes of value from the message source.
Incoming Message Queue Size	number (Range 0–max, Defaults 10)	Configures a maximum queue size for incoming messages. This is required to cope with rapid arrival of messages over the socket.

Property	Value	Description
Connection Attempt Timeout	number of milliseconds	Determines how long a station attempts to connect to a server before the attempt fails. This time should not be too short to cause false connection failures, and not so long as to cause excessive delays when a server is down.
Send Message Timeout	number of milliseconds	Configures the maximum amount of time to await for a message to be sent successfully.
Write on Start	`true` or `false`	Determines a writable proxy point's behavior when the station starts. `true` initiates a write when the station first reaches a steady state. `false` prevents a write when the station first reaches a steady state. **NOTE:** Consider setting to `false` except for critical proxy points, otherwise large networks may experience write-queue-overflow exceptions.
Write on Enabled	`true` or `false`	Determines a writable proxy point's behavior when the point's status transitions from disabled to normal (enabled). `true` initiates a write when the transition occurs. `false` prevents a write when the transition occurs.
Frame Buffer Size Bytes	number	Specifies the maximum size of each individual message frame.
Write Raw Bytes	`true` or `false`	Configures how the WebSocket client sends bytes. `true` sends message bytes as raw byte values. `false` sends message bytes as a WebSocket text messages.

Content Type Header (httpClient-ContentTypeHeader)

This component determines the content type and automatically loads it into a content-type header.

Figure 13 Content Type Header properties


lient : Request Body : Content Type Header
AX Property Sheet
Property Sheet
Content Type Header (Content Type Header)
Value
User Content Type
Refresh
Save

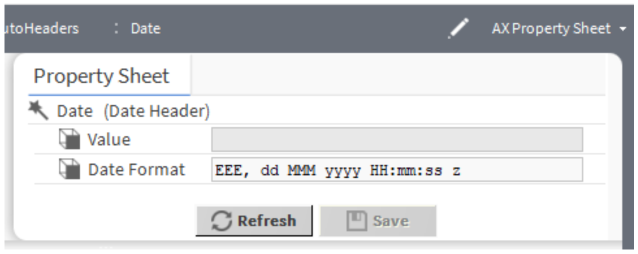
To access these properties, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice→Ping Address→Request Body** and double-click **Content Type Header**.

Properties	Value	Description
Value	read-only	Reports the calculated date.
User Content Type	text input field	Overrides the automatically calculated content type and offers auto-complete options.

Date (httpClient-DateHeader)

This component sets the date in a header.

Figure 14 Date properties



EEE, dd MMM yyyy HH:mm:ss z

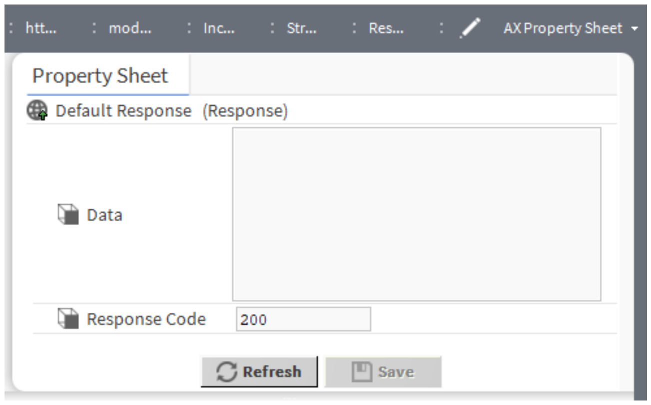
Drag an **AutoHeaders** component to the headers folder of a client, expand the component and double-click **Date**.

Property	Value	Description
Value	read only	Reports the calculated date.
Date Format	EEE, dd MMM yyyy HH:mm:ss z	Sets the date format for the header. This format displays the day of week, current date, current time and timezone.

Default Response (httpClient-Response)

This component configures the response to send back to the remote client.

Figure 15 Default Response properties


htt... : mod... : Inc... : Str... : Res...
AX Property Sheet
Property Sheet
Default Response (Response)
Data
Response Code
Refresh
Save
Response Code: 200

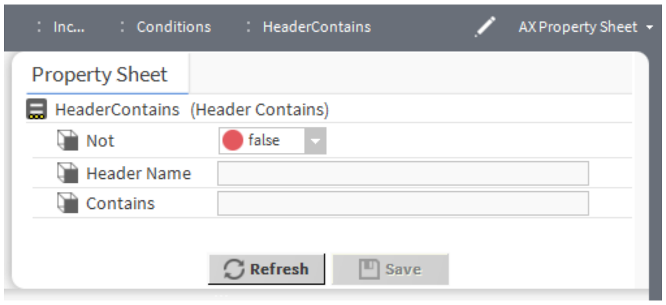
To access, drag a **Response** to a **Response Body** (**Response Folder**) under the **StringServlet** and double-click it.

Property	Value	Description
Data	text	Sets up the **Source** for this **Response Body**.
Response Code	number	Defines the HTTP status code for this response. These codes indicate if a specific HTTP request successfully completed or returned an error. Each number provides information about the request error.

Header Contains (httpClient-HeaderContains)

This component configures the header for a conditional response.

Figure 16 Header Contains properties



HeaderContains (Header Contains)

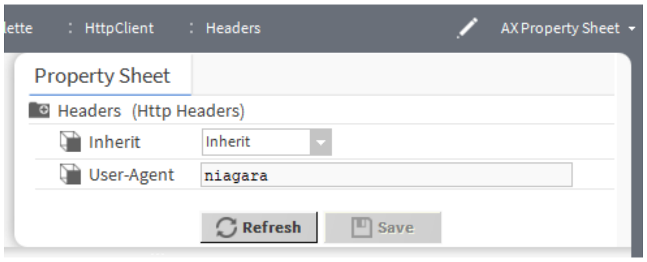
Not	● false
Header Name	
Contains	

Property	Value	Description
Not	`true` or `false` (default)	Indicates if the condition will be used (`false`) for the response or not (`true`).
Header Name	text	Specifies the name of the header in the request.
Contains	text	Defines a string to search for in the request body.

Headers (httpClient-HttpHeaders)

This component defines an access key, specifies your request's content type, and selects acceptable response content types. Unlike parameters, HTTP headers are not part of the address URL.

Figure 17 Headers properties



Headers (Http Headers)

Inherit	Inherit
User-Agent	niagara

To access these properties, double-click **HttpClient** and click **Headers**.

Property	Value	Description
Inherit	drop-down list	Determines the source of the header. `Inherit` merges header values defined within parent components, such as those in an **HttpClientFolder**, with a child component header. `Standalone` requires header values to be individually configured.
User-Agent	text	Provides a default user header value.

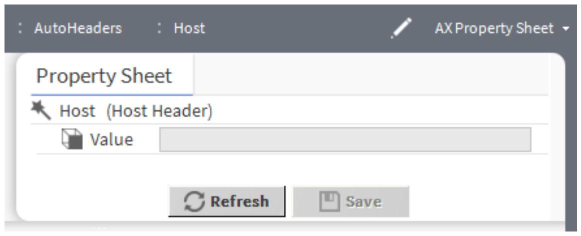
Action

Add adds a new header.

Host (httpClient-HostHeader)

This component defines the value for a header.

Figure 18 Host property



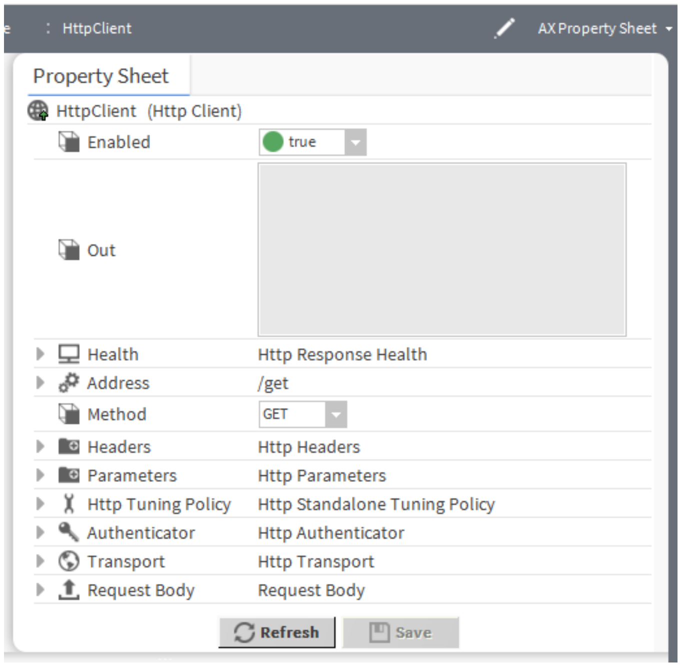
Drag an **AutoHeaders** component to the station, expand the component and double-click **Host**.

Property	Value	Description
Value	read-only	Reports the host name.

Http Client (httpClient-HttpClient)

This component is a standalone client, which you may use for making individual connections to single endpoints. You may use any type of request (GET/POST/PUT) with several configurations, such as parameters, headers and message body.

Figure 19 Http Client properties


: HttpClient
AX Property Sheet
Property Sheet
HttpClient (Http Client)

Enabled	● true
Out	
Health	Http Response Health
Address	/get
Method	GET
Headers	Http Headers
Parameters	Http Parameters
Http Tuning Policy	Http Standalone Tuning Policy
Authenticator	Http Authenticator
Transport	Http Transport
Request Body	Request Body

To access, drag this component to a location in the station, then double-click it in the station.

In addition to the standard properties (Enabled and Health), this component provides these properties.

Property	**Value**	**Description**
Out	text	Provides a current value, facets and status. • The value depends on the type of control point. • Facets, which define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. • The current status of the data item, meaning the health and validity of the value. Status is specified by a combination of status flags, such as `fault`, `overridden`, `alarm`, and so on. If no status flag is set, status is considered normal and reports `{ok}`.
Address	additional properties	Defines the address of the endpoint to which this client sends requests. For property descriptions refer to "Address (httpClient-HttpAddress)".
Method	drop-down list	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.

Property	Value	Description
Headers	additional properties	Contains additional information about an HTTP request or response sent between a client and server. For property descriptions refer to "Headers (httpClient-HttpHeaders)".
Parameters	additional properties	Contains key/value pair parameters for the request. For property descriptions refer to "Parameters (httpClient-HttpParameters)".
Http Tuning Policy	additional properties	Configures network rules for evaluating both write requests to writable proxy points as well as the acceptable freshness of read requests. For property descriptions refer to "Http Tuning Policy (httpClient-HttpStandaloneTuningPolicy)".
Authenticator	additional properties	Configures the authentication method. For property descriptions refer to "Authenticator (httpClient-HttpAuthenticator)".
Transport	additional properties	Configures the underlying transport layer. For property descriptions refer to "Transport (httpClient-HttpTransport)".
Request Body	additional properties	Configures the request body content. For property descriptions refer to "Request Body (httpClient-HttpRequestBody)".

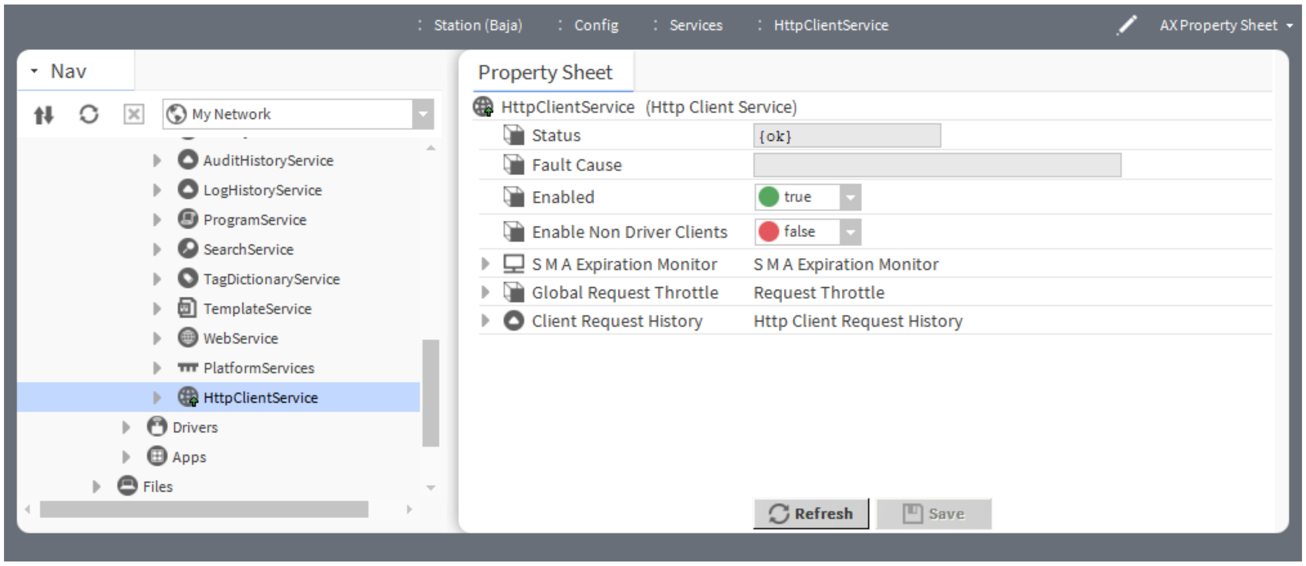
Actions

- **Send** sends the selected request.
- **Clear Last Result** clears the previous result.
- **Add More** adds more clients.

Http Client Service (httpClient-HttpClientService)

This component automatically appears in your **Services** container, when you add an HTTP Client to a running station. This includes an SMA expiration monitor for configuration of alarms that reports when the stations maintenance agreement is close to expiry.

Figure 20 Http Client Service properties



HttpClientService (Http Client Service)

Status	{ok}
Fault Cause	
Enabled	true
Enable Non Driver Clients	false
S M A Expiration Monitor	S M A Expiration Monitor
Global Request Throttle	Request Throttle
Client Request History	Http Client Request History

To access, expand **Config→Services** and double-click **HttpClientService**.

In addition to the standard properties (Status, Enabled and Fault Cause), this component includes a single slot.

Property	Value	Description
Enable Non Driver Clients	`true` or `false` (default)	Enables (`true`) and disables (`false`) non-driver client types, such as Standalone Http Client and WebSocket Client.
SMA Expiration Monitor	additional properties	Configures a reminder of when the framework Software Maintenance Agreement is about to expire. For property descriptions, refer to "S M A Expiration Monitor (httpClient-SMAExpirationMonitor)".
Global Request Throttle	additional properties	Allows a global limit on all outgoing client requests within a configured timeframe. For property descriptions, refer to "Request Throttle (httpClient-HttpRequestThrottle)".
Client Request History	additional properties	Logs the most recent http client requests in an audit history named "HttpClientRequestHistory." For property descriptions, refer to "Http Client Request History (httpClient-ClientRequestHistory)".

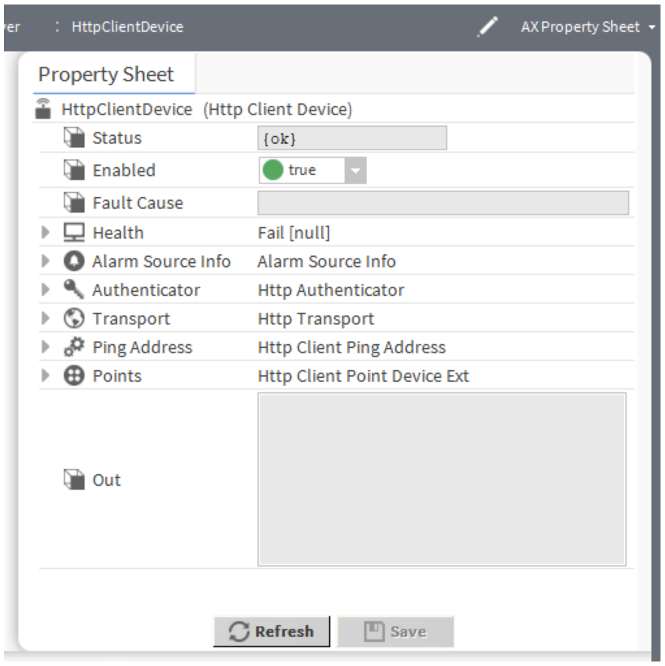
Actions

- **Enable All** enables all the http clients to access the service.
- **Disable All** disables all the http clients from accessing the service.

Http Client Device (httpClient-HttpClientDevice)

This component configures a client device.

Figure 21 Http Client Device properties


er : HttpClientDevice AX Property Sheet

Property Sheet

HttpClientDevice (Http Client Device)

Status	{ok}
Enabled	true
Fault Cause	
Health	Fail [null]
Alarm Source Info	Alarm Source Info
Authenticator	Http Authenticator
Transport	Http Transport
Ping Address	Http Client Ping Address
Points	Http Client Point Device Ext
Out	

To access, expand **Config→Drivers→HttpClientNetwork** and double-click **HttpClientDevice**.

In addition to the standard properties (Status, Enabled, Health and Alarm Source Info), this component provides these properties

Property	Value	Description
Authenticator	additional properties	Configures the authentication method. For property details, refer to "Authenticator (httpClient-HttpAuthenticator)".
Transport	additional properties	Configures the underlying transport layer. For property details, refer to "Transport (httpClient-HttpTransport)".
Ping Address	additional properties	Configures a device status ping scan by connecting to a URL over HTTP and reading the HTTP response. For property details, refer to "Ping Address (httpClient-HttpClientPingAddress)".
Points	container	Serves as a container for HTTP client points.
Out	read-only	Provides a current value, facets and status. • The value depends on the type of control point. • Facets, which define how the value displays, including the value's number of decimal places, engineering units, or text descriptors for Boolean/enum states. • The current status of the data item, meaning the health and validity of the value. Status is specified by a combination of status flags, such as `fault`, `overridden`, `alarm`, and so on. If no status flag is set, status is considered normal and reports `{ok}`.

Action

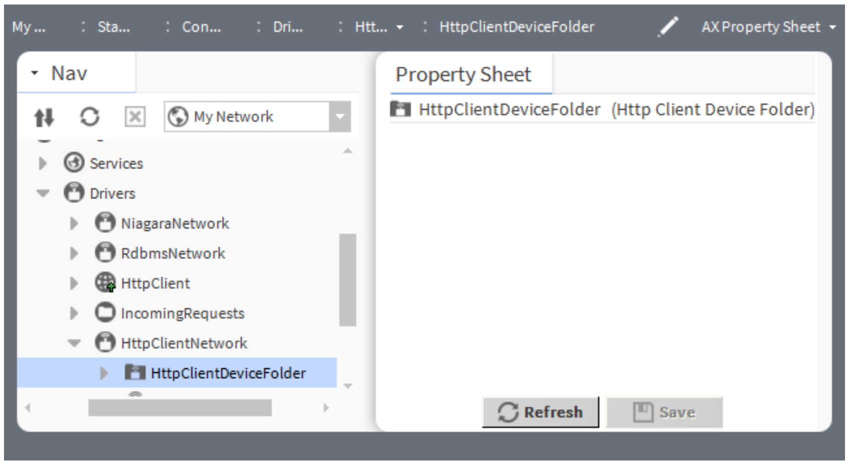
Ping sends a message to a URL. The message provokes a response, which indicates the current state of the object.

Http Client Device Folder (httpClient-HttpClientDeviceFolder)

This folder component organises devices under the network component.

The default view for this component is the **Http Client Device Manager**.

Figure 22 Http Client Device Folder



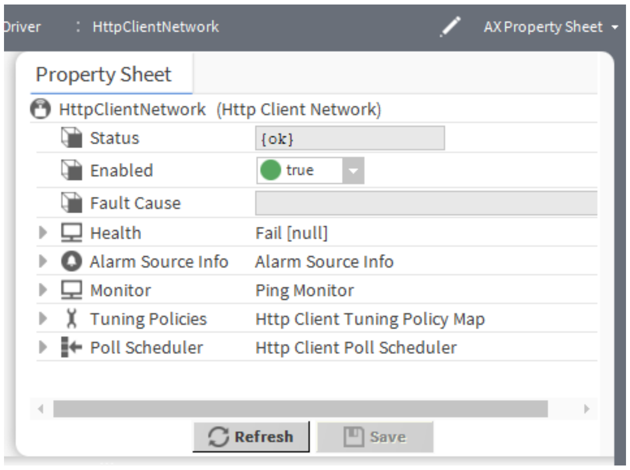
HttpClientDeviceFolder (Http Client Device Folder)

To access, expand **Config→Drivers→HttpClientNetwork→HttpClientDeviceFolder** and click **Views→AX Property Sheet**.

Http Client Network (httpClient-HttpClientNetwork)

This component configures the **HttpClientNetwork**, which offers the same functionality as a standalone client with the addition of several related endpoints. These endpoints serve as child **StringPoint** components with configurable proxy extensions per request. Each request can have a different address and a different set of parameters, headers and message body.

Figure 23 Http Client Network properties



HttpClientNetwork (Http Client Network)

Status	{ok}
Enabled	true
Fault Cause	
Health	Fail [null]
Alarm Source Info	Alarm Source Info
Monitor	Ping Monitor
Tuning Policies	Http Client Tuning Policy Map
Poll Scheduler	Http Client Poll Scheduler

To access, expand **Config→Drivers** , right-click **HttpClientNetwork** and click **Views** > **AX Property Sheet**.

All these properties are standard network properties, which are documented in the *Niagara Drivers Guide*

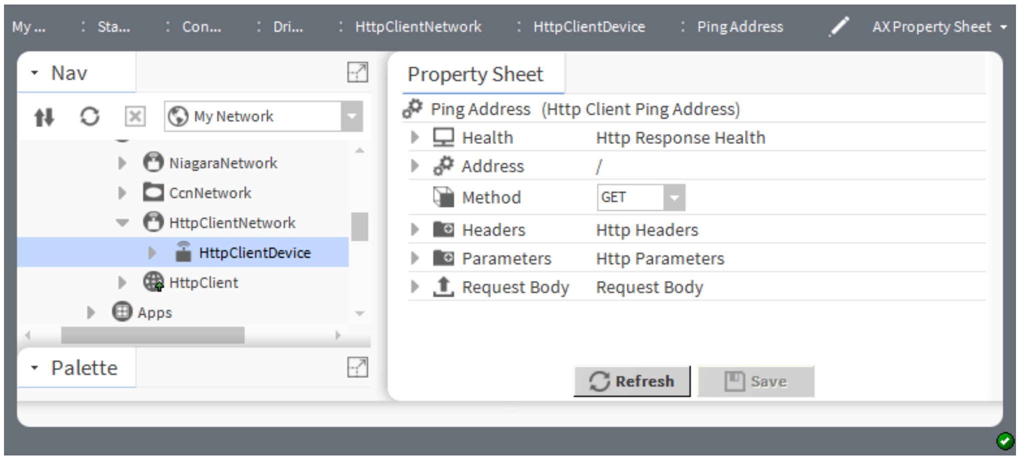
Action

Ping sends a message to a network object (device, database, etc). The message provokes a response, which indicates the current state of the object.

Http Client Ping Address (httpClient-HttpClientPingAddress)

This component configures a device status ping.

Figure 24 Http Client Ping Address properties



Ping Address (Http Client Ping Address)

Health	Http Response Health
Address	/
Method	GET
Headers	Http Headers
Parameters	Http Parameters
Request Body	Request Body

To access these properties, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice** and double-click **Ping Address**.

In addition to the standard property, `Health`, these properties support this component.

Property	Value	Description
Address	additional properties	Defines the address of the endpoint to which this client sends requests. For property descriptions, refer to "Address (httpClient-HttpAddress)".
Method	drop-down list	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.
Headers	additional properties	Contains additional information about an HTTP request or response sent between a client and server. For property descriptions refer to "Headers (httpClient-HttpHeaders)".

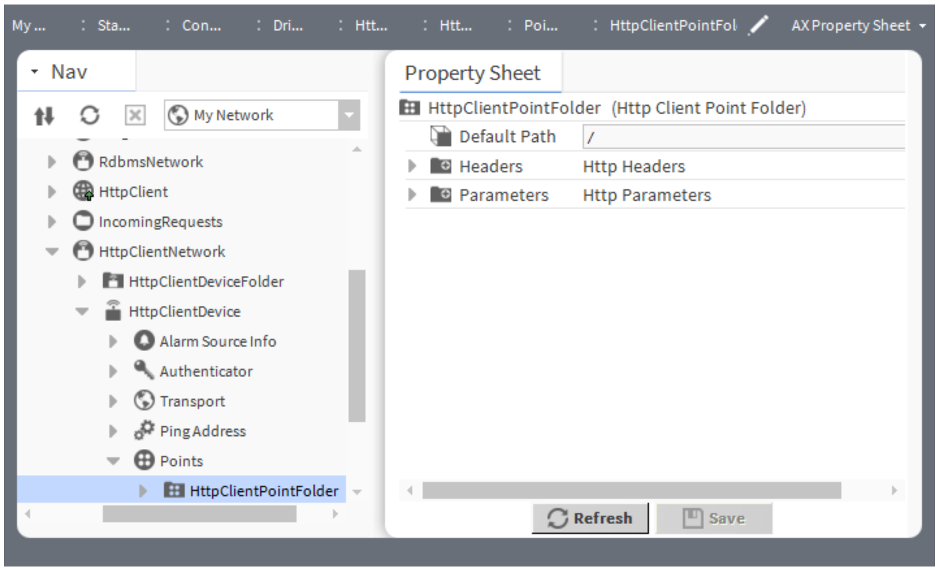
Property	Value	Description
Parameters	additional properties	Contains key/value pair parameters for the request. For property descriptions refer to "Parameters (httpClient-HttpParameters)".
Request Body	additional properties	Configures the request body content. For property descriptions refer to "Response Body (httpClient-ResponseFolder)".

Http Client Point Folder (httpClient-HttpClientPointFolder)

This folder component organises HTTP points under the client device.

The default view for this component is the **Http Client Point Manager**.

Figure 25 Http Client Point Folder properties



HttpClientPointFolder (Http Client Point Folder)

Default Path	/
Headers	Http Headers
Parameters	Http Parameters

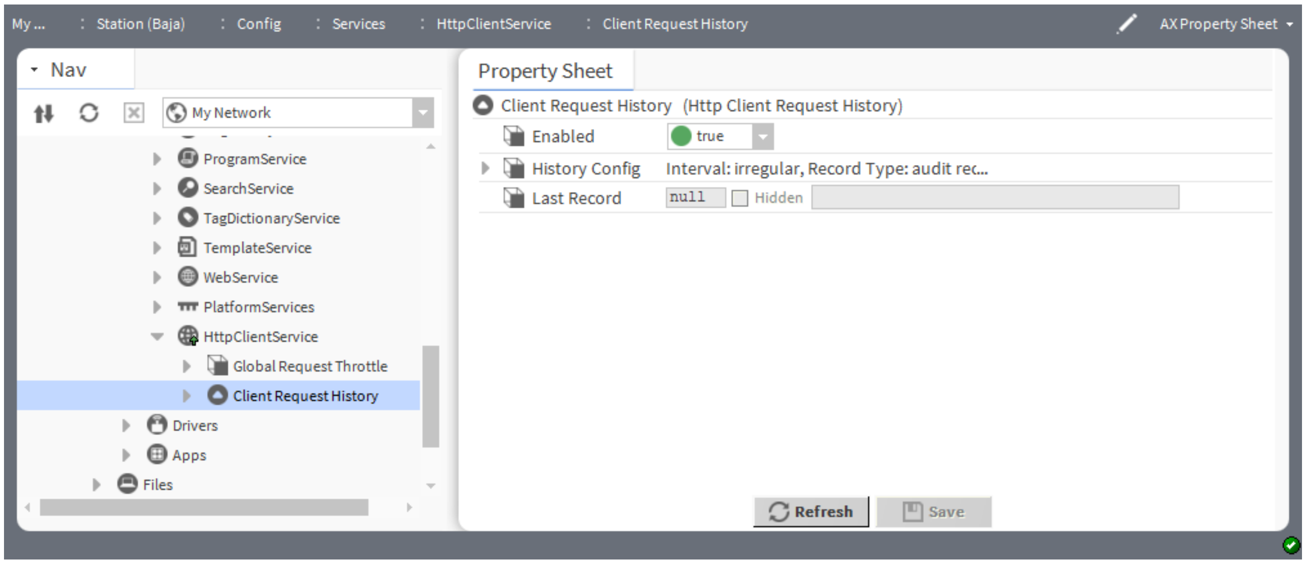
To access, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice→Points** and right click **HttpClientPointFolder→Views→AX Property Sheet**.

Property	Value	Description
Default Path	text	Defines the default path for child points, which have the `Inherit` property.
Headers	additional properties	Defines the default headers for child points, which have the `Inherit` property. For property details, refer to "Headers (httpClient-HttpHeaders)".
Parameters	additional properties	Defines the default parameters for child points, which have the `Inherit` property. For property details, refer to "Parameters (httpClient-HttpParameters)".

Http Client Request History (httpClient-ClientRequestHistory)

This component logs the most recent HTTP client requests in an audit history named "HttpClientRequestHistory"'.

Figure 26 Http Client Request History properties



Client Request History (Http Client Request History)
Enabled true
History Config Interval: irregular, Record Type: audit rec...
Last Record null Hidden

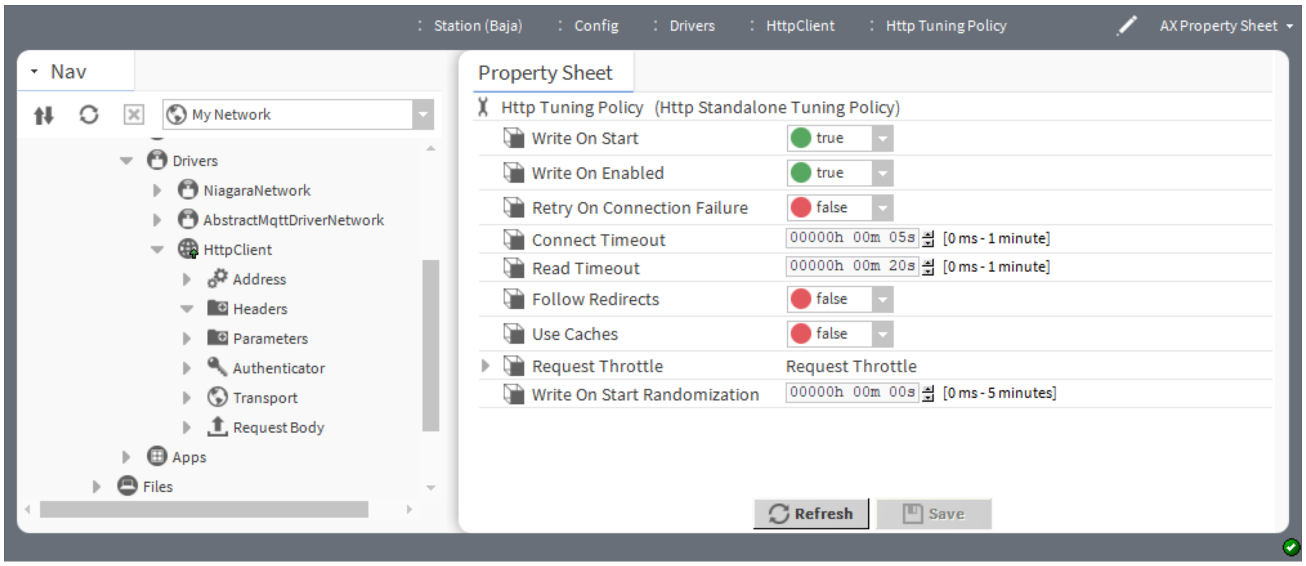
To access, expand **Config→Services→HttpClientService** and double-click **ClientRequestHistory**.

Property	**Value**	**Description**
Enabled	`true` or `false` (default)	Enables (`true`) and disables (`false`) this request throttle.
History Config	additional properties	Allows the history to be disabled, or to change the amount of client requests logged. For property descriptions, refer to *Niagara Histories Guide* "history-HistoryConfig".
Last Record	read only	Displays the last history record.

Http Tuning Policy (httpClient-Http StandaloneTuning Policy)

This component configures the network's rules for evaluating both write requests as well as the acceptable freshness of read requests that result from polling.

Figure 27 Http Tuning Policy properties



Http Tuning Policy (Http Standalone Tuning Policy)

Write On Start	🟢 true	
Write On Enabled	🟢 true	
Retry On Connection Failure	🔴 false	
Connect Timeout	00000h 00m 05s	[0 ms - 1 minute]
Read Timeout	00000h 00m 20s	[0 ms - 1 minute]
Follow Redirects	🔴 false	
Use Caches	🔴 false	
Request Throttle	Request Throttle	
Write On Start Randomization	00000h 00m 00s	[0 ms - 5 minutes]

To access these properties, double-click **HttpClient** and double-click **Http Tuning Policy**.

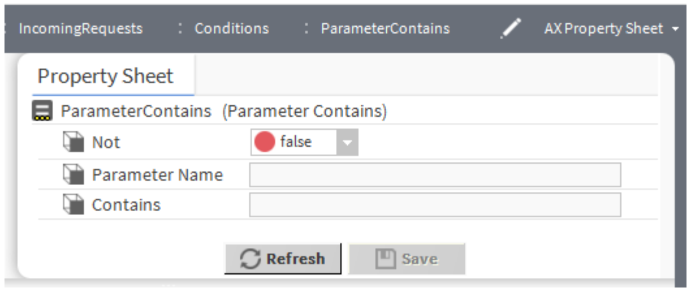
Property	Value	Description
Write On Start	`true` (default) or `false`	Determines a writable proxy point's behavior when the station starts. `true` initiates a write when the station first reaches a steady state. `false` prevents a write when the station first reaches a steady state. **NOTE:** Consider setting to `false` except for critical proxy points, otherwise large networks may experience write-queue-overflow exceptions.
Write On Enabled	`true` (default) or `false`	Determines a writable proxy point's behavior when the point's status transitions from disabled to normal (enabled). `true` initiates a write when the transition occurs. `false` prevents a write when the transition occurs.
Retry On Connection Failure	`true` (default) or `false`	Configures what happens if the connection fails. `true` makes a single retry attempt. `false` does not retry the connection.
Connect Timeout	number of milliseconds	Determines how long a station attempts to connect to a server before the attempt fails. This time should not be too short to cause false connection failures, and not so long as to cause excessive delays when a server is down.
Read Timeout	number of milliseconds	Defines the maximum amount of time to wait for a response to a read.
Follow Redirects	`true` (default) or `false`	Move content to a new URL. `true`, automatically follows any 302 responses to the new address.

Property	Value	Description
		`false` does nothing with a redirect.
Use Caches	`true` (default) or `false`	Controls the Cache-Control http header. `true` enables the outgoing Cache-Control http header. `false` disables cache.
Request Throttle	additional properties	Allows a limit on outgoing requests for this client within a configured timeframe. For property descriptions, refer to "Request Throttle (httpClient-HttpRequestThrottle)".
Write On Start Randomization	number of milliseconds	Selects a random maximum number of seconds after the station starts before commencing a send.

Parameter Contains (httpClient-ParameterContains)

This component configures parameter conditions.

Figure 28 Parameter Contains properties



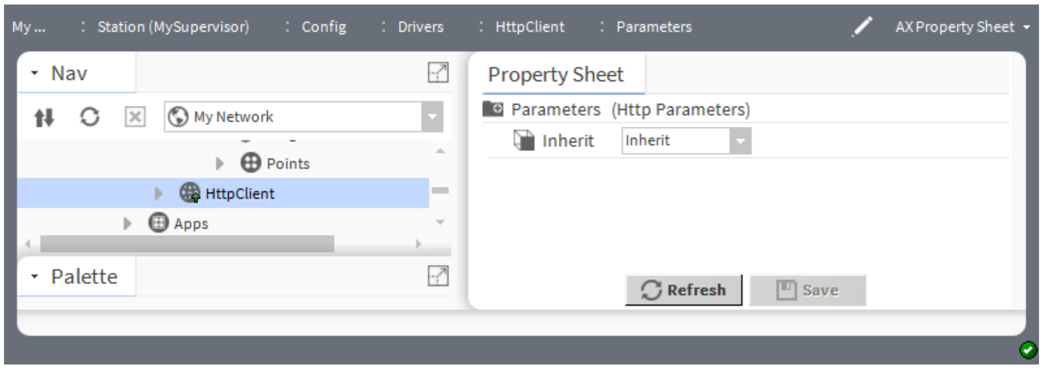
To access these properties, expand **Config→Drivers→IncomingRequests→StringServlet→Conditions** and double-click **ParameterContains**.

Property	Value	Description
Not	`true` or `false` (default)	Indicates if the condition will be used (`false`) for the response or not (`true`).
Parameter Name	text	Specifies the name of the parameter for the request.
Contains	text	Defines a string to search for in the request body.

Parameters (HttpClient-HttpParameters)

This component configures a single property for HTTP parameters.

Figure 29 Parameters property



Parameters (Http Parameters)
Inherit Inherit

To access these properties, double-click **HttpClient** and double-click **Parameters**.

Property	Value	Description
Inherit	drop-down list	Determines the source of the parameter. `Inherit` merges parameter values defined within parent components, such as those in an **HttpClientFolder**, with a child component parameter. `Standalone` requires parameter values to be individually configured.

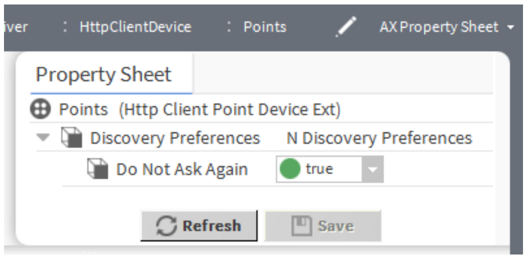
Action

Add adds a new parameter.

Points (httpClient-HttpClientPointDeviceExt)

This component is an implementation of a **PointDeviceExt**. Its primary view is the **Point Manager**.

Figure 30 Points property



Points (Http Client Point Device Ext)
Discovery Preferences N Discovery Preferences
Do Not Ask Again true

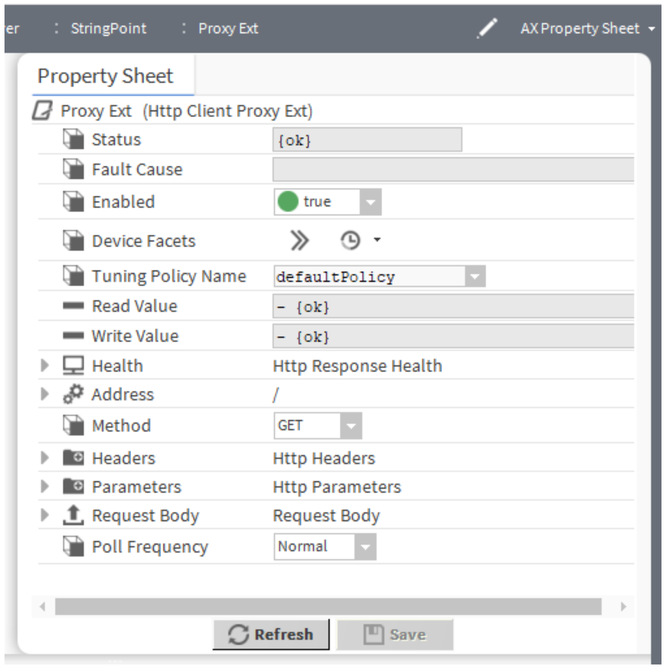
To access, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice**, right-click **Points**, click **Views→AX Property Sheet** and expand **Discovery Preferences**.

Property	Value	Description
Do Not Ask Again	`true` (default) or `false`	Hides (`true`) the Discovery window (prompt) that normally opens when you click the **Discover** button on the **Device Manager** view. `false` allows the window to open before the system initiates the discovery search.

Proxy Ext (httpClient-HttpClientProxyExt)

This component contains all of the features of the standalone client.

Figure 31 Proxy Ext properties



Proxy Ext (Http Client Proxy Ext)

Status	{ok}
Fault Cause	
Enabled	true
Device Facets	
Tuning Policy Name	defaultPolicy
Read Value	- {ok}
Write Value	- {ok}
Health	Http Response Health
Address	/
Method	GET
Headers	Http Headers
Parameters	Http Parameters
Request Body	Request Body
Poll Frequency	Normal

To access, expand **Config→Drivers→HttpClientNetwork→StringPoint** and expand or click **ProxyExt**.

In addition to the standard properties (Status, Enabled, Fault Cause, Device facets, Tuning Policy Name, Health, Read Value, Write Value and Poll Frequency), this component provides these properties.

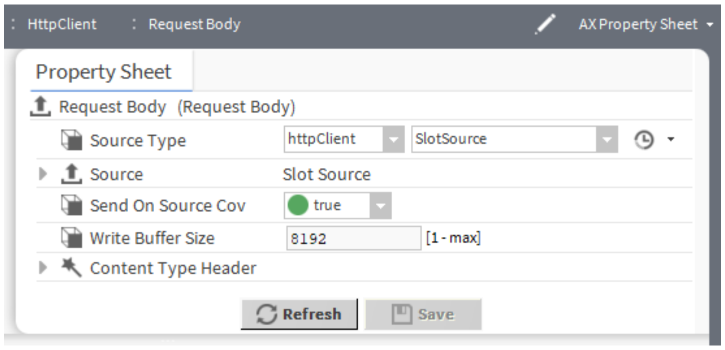
Property	Value	Description
Address	additional properties	Defines the address of the endpoint to which this client sends requests. For property descriptions refer to "Address (httpClient-HttpAddress)".
Method	drop-down list	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.
Headers	additional properties	Contains additional information about an HTTP request or response sent between a client and server. For property descriptions refer to "Headers (httpClient-HttpHeaders)".

Property	Value	Description
Parameters	additional properties	Contains key/value pair parameters for the request. For property descriptions refer to "Parameters (httpClient-HttpParameters)".
Request Body	additional properties	Configures the request body content. For property descriptions refer to "Response Body (httpClient-ResponseFolder)".

Request Body (httpClient-RequestBody)

This component configures the source type and the **Send** action properties of a request body.

Figure 32 httpClient-RequestBody properties



Request Body (Request Body)

Source Type	httpClient	SlotSource
Source	Slot Source	
Send On Source Cov	true	
Write Buffer Size	8192	[1 - max]
Content Type Header		

To access these properties, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice→Ping Address** and double-click **Request Body**.

Properties	Value	Description

Source Type	drop-down list (defaults to `SlotSource`)	Selects the source type for sending the content.
		`FileSource`
		`ParameterStringSource`
		`ReportPayloadSource`
		`SlotSource`

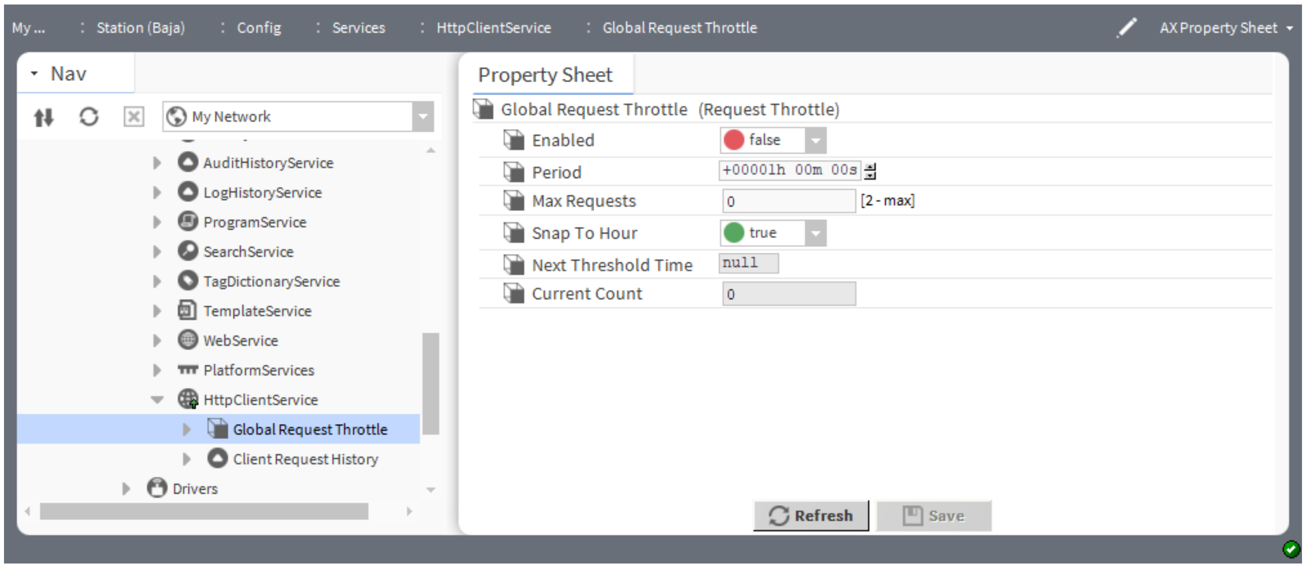
Clear Payload After	drop-down list (defaults to `Neither`)	Selects the status required to clear a payload.
		`Success`
		`Failure`
		`Neither`
		`Both`

Properties	Value	Description
Send On Source Cov	`true` (default) or `false`	Specifies if a new Http request is automatically sent after modifying the **`Data Slot`** value in the source. `true` sends a new Http request `false` causes the **Send** action to occur only when executed. By default, when you modify the **`Data`** slot value in the request body source, the driver automatically sends a new HTTP request. To alter this behaviour, set **`Send On Source Cov`** to `false` under **`Request Body`**. Then send only occurs when you invoke the **Send** action.
Write Buffer Size	number (defaults to 8192)	Specifies the size of the buffer to use when reading the source data into the Http connection for tuning. A higher value may increase the performance for large message bodies.

Request Throttle (httpClient-HttpRequestThrottle)

Allows a limit to be configured on the number of outgoing httpClient requests within a timeframe. This is useful for preventing accidental spamming of a remote service, or to ensure your traffic remains in the terms of use for an api service.

Figure 33 Request Throttle properties



Global Request Throttle (Request Throttle)

Enabled	false
Period	+00001h 00m 00s
Max Requests	0 [2 - max]
Snap To Hour	true
Next Threshold Time	null
Current Count	0

To access, expand **Config→Services→HttpClientService** and double-click **GlobalRequestThrottle**.

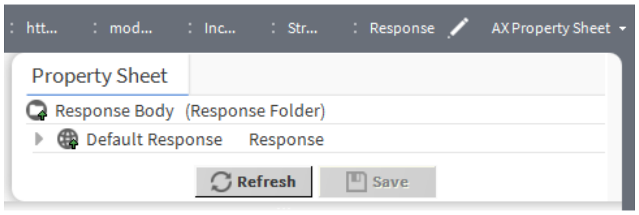
Property	Value	Description
Enabled	`true` or `false` (default)	Enables (`true`) and disables (`false`) this request throttle.
Period	number of minutes	Configures the length of time to apply to the **`Max Requests`** threshold. The software sets this period when the first request is made, and recalculates it on the first request after expiry.
Max Requests	number	Sets up the maximum permitted number of requests within the period. Any requests exceeding this total result in a failed request send attempt.

Property	Value	Description
Snap To Hour	`true` (default) or `false`	Configures (`true`) the next period to start at the next hour. This removes all minutes and seconds to end the current period at the start of the next hour. `false` allows the next period to cross the start of the next hour.
Next Threshold Time	read only	Reports the end of the current request period.
Current Count	read only	Reports how many requests have occurred in the current request period.

Response Body (httpClient-ResponseFolder)

This component contains a response.

Figure 34 Response Body property



Response Body (Response Folder)
Default Response Response

To access these properties, expand **ConfigDriversIncomingRequestsStringServlet**, right-click **Response Body** and click **View** > **AX Property Sheet**.

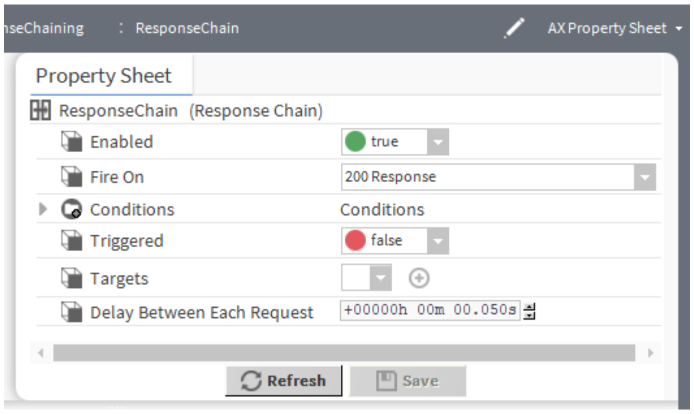
Property	Value	Description
Default Response	additional properties	Contains the default response and code. For property details, refer to "Default Response (httpClient-Response)".

Response Chain (httpClient-ResponseChain)

This component is functionally the same as the **ResponseTrigger** component, with two additional properties (**`Targets`** and **`Delay Between Each Request`**), which cause one or more secondary HTTP client components to send when the trigger logic fires.

You add this component to either a HttpClient component, or to the point's **Proxy Ext** (**HttpClientNetwork→HttpClientDevice→Points→StringPoint** in the Nav tree. The **ResponseChain** evaluates its logic each time its parent receives a response.

Figure 35 Response Chain properties



ResponseChain (Response Chain)

Enabled	true
Fire On	200 Response
Conditions	Conditions
Triggered	false
Targets	
Delay Between Each Request	+00000h 00m 00.050s

To access, expand **Config→Drivers→HttpClient→ResponseChain** and right click **Views→AX Property Sheet.**

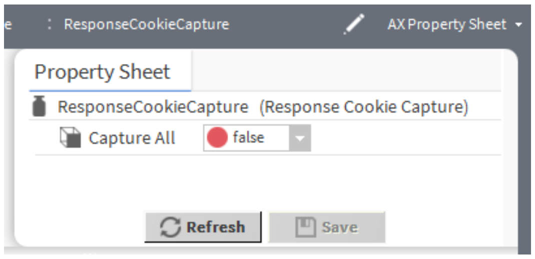
In addition to the standard property (Enabled), this component provides these properties.

Property	Value	Description
Fire On	drop-down list	Defines the trigger criterion using a response code: `200 Response` defines a specific response code. `On2xx` (200 -299) provides a response code range. `Unauthorized/Forbidden` provides an unauthorized (401) or forbidden (403) response. `Response Code Changed from previous` defines any code that is different from the previous code. `All Responses` defines any code.
Conditions	additional properties	Provides a second way to define a trigger criterion by adding one or two conditions from the palette (**BodyContains** and **HeaderContains**), then configures the **`And`** Boolean property appropriately. For property descriptions refer to “Conditions (httpClient-HttpConditions)”.
Triggered	read-only	Reports `true` when the **ResponseTrigger** component’s logical criteria have been fulfilled. Otherwise, it reports (`false`). `true` also fires the trigger topic when the logical criterion has been fulfilled. You may link either of these slots to **Wire Sheet** logic.
Targets	drop-down list	Selects one or more secondary clients to add (⊕) to the list of targets.
Delay Between Each Request	hours minutes seconds	Defines the minimum amount of time to elapse between the invocation of the **Send** action for each target client.

Response Cookie Capture (httpClient-ResponseCookieCapture)

This component captures cookie values from a response for the purpose of linking within **Wire Sheet** logic, or to use as a cookie value in another client request.

Figure 36 Response Cookie Capture property



ResponseCookieCapture (Response Cookie Capture)
Capture All false

Drag one of these components into the station and double-click it.

Property	Value	Description
Capture All	`true` or `false` (default)	Specifies what the response does when capturing cookies. `true` creates or updates all received cookies. `false` does not create new cookies but updates the existing cookies.

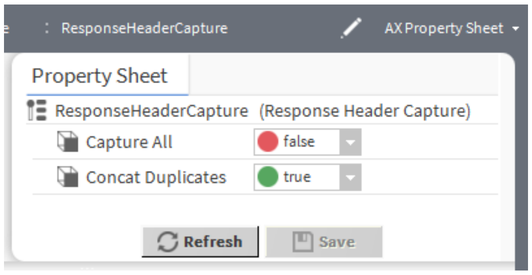
Actions

- **Add** adds a new header.
- **Clear All** clears all the headers.
- **Reset** returns all header properties to their original values.

Response Header Capture (httpClient-ResponseHeaderCapture)

This component captures headers from a response for the purpose of linking within Wire Sheet logic, or to use as a header value on another client request.

Figure 37 Response Header Capture properties



ResponseHeaderCapture (Response Header Capture)
Capture All false
Concat Duplicates true

Drag one of these components into the station and double-click it.

Property	Value	Description
Capture All	`true` or `false` (default)	Specifies what the response does when capturing headers. `true` creates or updates all received headers. `false` does not create new headers but updates existing headers.
Concat Duplicates	`true` (default)or `false`	Configures what happens if a response contains two headers with the same name. `true` combines the two headers in a response if the headers contain the same name, and concatenates their values as a CSV string. `false` does not combine the headers and does not concatenate their values as a CSV string.

Actions

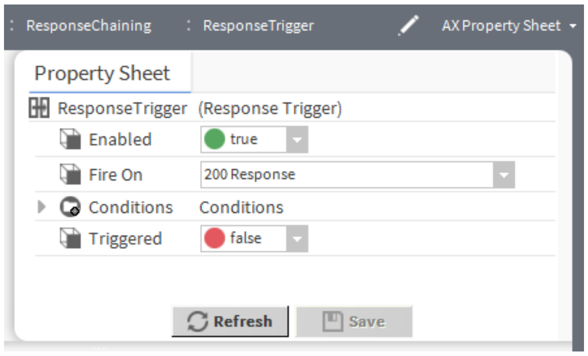
- **Add** adds a new header.
- **Clear All** clears all the headers.
- **Reset** returns all header properties to their original values.

Response Trigger (httpClient-ResponseTrigger)

This component triggers events or secondary client requests after an initial **HttpClient** request has completed.

You add a **ResponseTrigger** to either a **HttpClient** component or **Http Client Proxt Ext** (this Proxy Ext is under **HttpClientNetwork→HttpClientDevice→Points→StringPoint**). The **ResponseTrigger** evaluates its logic each time the parent receives a response.

Figure 38 Response Trigger properties



ResponseTrigger (Response Trigger)

Enabled	● true
Fire On	200 Response
Conditions	Conditions
Triggered	● false

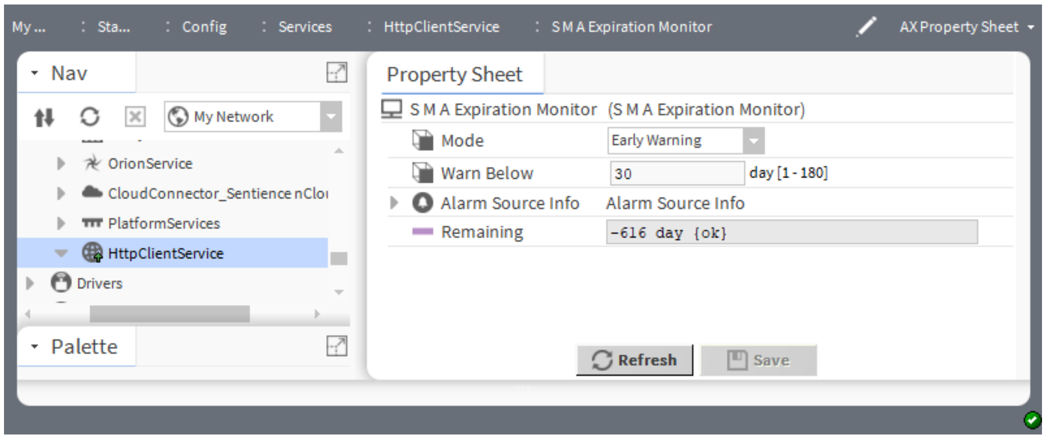
In addition to the standard property (Enabled), this component provides these properties.

Property	Value	Description
Fire On	drop-down list	Defines the trigger criterion using a response code: `200 Response` defines a specific response code. `On2xx` (200 -299) provides a response code range. `Unauthorized/Forbidden` provides an unauthorized (401) or forbidden (403) response. `Response Code Changed from previous` defines any code that is different from the previous code. `All Responses` defines any code.
Conditions	additional properties	Provides a second way to define a trigger criterion by adding one or two conditions from the palette (**BodyContains** and **HeaderContains**), then configures the **And** Boolean property appropriately. For property descriptions refer to "Conditions (httpClient-HttpConditions)".
Triggered	`true` or `false` (default)	Reports `true` when the **ResponseTrigger** component's logical criteria have been fulfilled. Otherwise, it reports (`false`). `true` also fires the trigger topic when the logical criterion has been fulfilled. You may link either of these slots to **Wire Sheet** logic.

S M A Expiration Monitor (httpClient-SMAExpirationMonitor)

This component configures alarms to report when the stations maintenance agreement is close to expiry.

Figure 39 S M A Expiration Monitor properties



S M A Expiration Monitor (S M A Expiration Monitor)
Mode: Early Warning
Warn Below: 30 day [1 - 180]
Alarm Source Info: Alarm Source Info
Remaining: -616 day {ok}

To access, expand **Config→Services**, double-click **HttpClientService** and click **S M A Expiration Monitor**.

In addition to standard component Alarm Source info, these properties are unique to the S M A Expiration Monitor.

Property	Value	Description
Mode	drop-down list (defaults to `Early Warning`)	Configures when to activate an alarm regarding a pending license expiration. `Early Warning`: generates an alarm before the license expires. `Once Expired`: generates an alarm when the license expires and thereafter. `Disable Monitor`: turns monitoring off.
Warn Below	number of days from 1 to 180 (defaults to 30 days)	Configures when to start warning of the license expiration.
Remaining	read-only	Displays the number of days before the license expires.

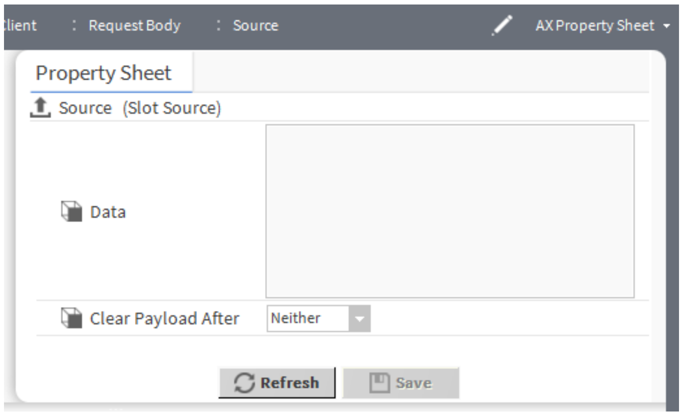
Action

Check Maintenance Expiration updates the **`Remaining`** value.

Source (httpClient-SlotSource)

This component provides an additional message to request or update data within a resource.

Figure 40 Source properties



To access these properties, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice→Ping Address→Request Body** and double-click **Source**.

Properties	Value	Description
Data	message input field	Sets up the **`Source`** for this **`Response Body`**.

Clear Payload After	drop-down list (defaults to `Neither`)	Selects the status required to clear a payload.
		`Success`
		`Failure`
		`Neither`
		`Both`

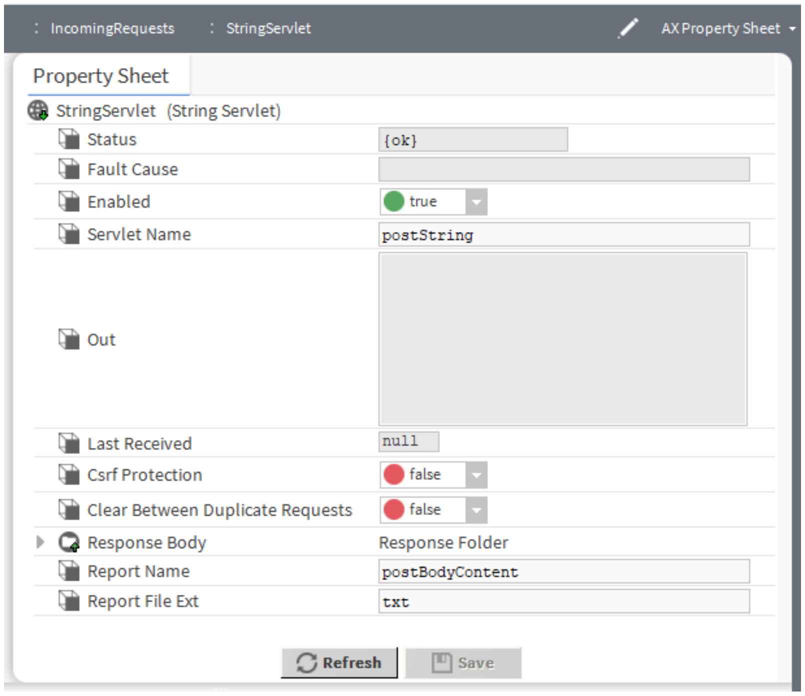
StringServlet (httpClient-StringServlet)

This component captures requests coming in to the station from an external client. You can also send GET requests to a **StringServlet**. The functionality is the same, except no **Request Body** can be posted.

This may be any Http client such as:

- a web browser
- a command line utility, such as curl or wget
- an application for creating requests, such as Postman
- another **httpClient** instance running on another station

Figure 41 StringServlet properties



StringServlet (String Servlet)

Status	{ok}
Fault Cause	
Enabled	true
Servlet Name	postString
Out	
Last Received	null
Csrf Protection	false
Clear Between Duplicate Requests	false
Response Body	Response Folder
Report Name	postBodyContent
Report File Ext	txt

To access, expand **ConfigDriversIncomingRequests** and drag this component to **IncomingRequests**, then double-click it.

In addition to the standard properties (Status, Enabled, and Fault Cause), this component provides these properties.

Property	**Value**	**Description**
Servlet Name	text	Defines the name of the servlet.
Out	read-only	Displays the message body of any POST request, when an HTTP request is sent to the **StringServlet**.
Last Received	read-only	Displays when the the last message was received.
Csrf Protection	`true` or `false` (default)	Turns CSRF protection on and off. `true` enables CSRF protection. `false` disables CSRF protection.

Property	Value	Description
Clear Between Duplicate Request	`true` or `false` (default)	Configures what happens between duplicate requests. `true` clears messages between duplicate requests. `false` disables this function.
Response Body	additional properties	Configures the request body content. For property descriptions refer to "Request Body (httpClient-HttpRequestBody)".
Report Name	text	Defines the name of the report for an incoming request.
Report File Ext	text	Defines the file extension of the report file.

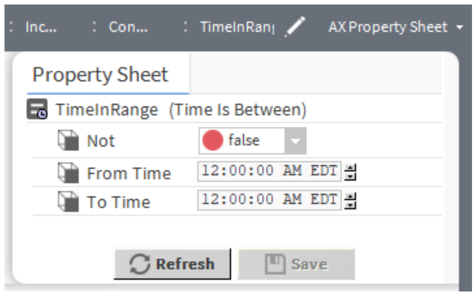
Action

Reset returns all properties to their original values.

Time Is Between (httpClient-TimeIsBetween)

This Time In Range component configures time-related properties.

Figure 42 Time Is Between properties



TimeInRange (Time Is Between)
Not false
From Time 12:00:00 AM EDT
To Time 12:00:00 AM EDT

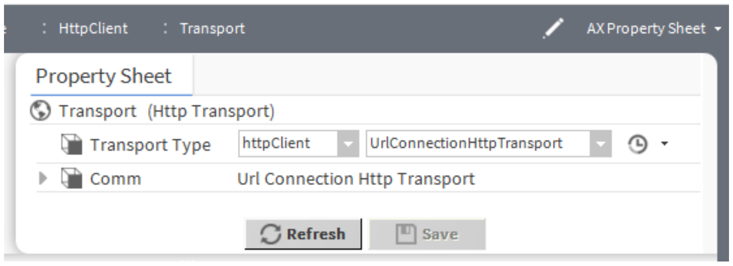
Property	Value	Description
Not	`true` or `false` (default)	Indicates if the condition will be used (`false`) for the response or not (`true`).
From Time	hours, minutes, seconds	Specifies when this condition becomes active.
To Time	hours, minutes, seconds	Specifies when this condition ceases to be active.

Transport (httpClient-Http Transport)

This component switches an underlying Http client transport layer between that which comes with the standard JRE and the third party OKHttp library. This allows the module to potentially work around behaviors seen with either implementation by providing a choice.

Both the Http Client driver and the standalone clients contain a transport selector.

Figure 43 Transport properties



Transport (Http Transport)

Transport Type	httpClient UrlConnectionHttpTransport
Comm	Url Connection Http Transport

To access these properties, double-click **HttpClient** and double-click **Transport**.

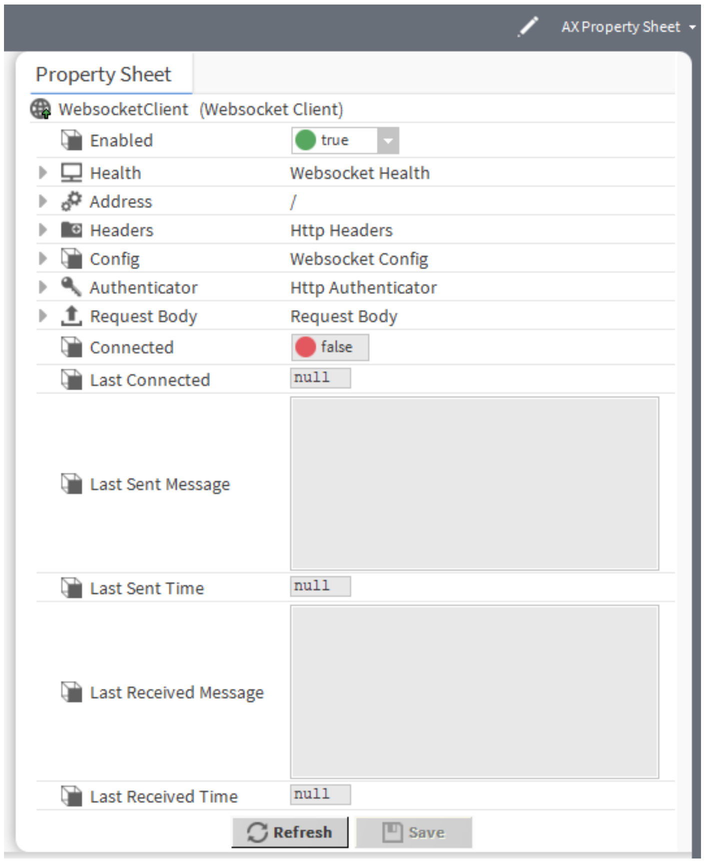
Property	Value	Description
Transport Type	drop-down list	Switches the underlying transport layer between the standard JRE (`UrlConnectionHttpTransport`) and the third-party OKHttp library (`OKHttp Transport`).

Websocket Client (httpClient-WebsocketClient)

This component has similar functionality to the standalone http client component. A WebSocket is a persistent connection to an endpoint allowing full-duplex communications, where either the client or server side sends a message at any time.

The **WebSocketClient** contains many configuration features similar to the HTTP client components, such as an **Address**, a **Headers** folder, **`Authenticator`** and **`Request Body`** for defining the message content. The main difference is that the **WebSocketClient** has a **`Connected`** property to indicate whether the persistent connection is currently active, and slots to hold both the last sent and received messages.

Figure 44 Websocket Client properties



WebsocketClient (Websocket Client)	
Enabled	● true
Health	Websocket Health
Address	/
Headers	Http Headers
Config	Websocket Config
Authenticator	Http Authenticator
Request Body	Request Body
Connected	● false
Last Connected	null
Last Sent Message	
Last Sent Time	null
Last Received Message	
Last Received Time	null

To access these properties, drag this component to a location in the station, then double-click it in the station.

In addition to the standard properties (Enabled and Health), this component provides these properties.

Property	Value	Description
Address	additional properties	Defines the address of the endpoint to which this client sends requests. For property descriptions, refer to “Address (httpClient-HttpAddress)”.
Headers	additional properties	Contains additional information about an HTTP request or response sent between a client and server. For property descriptions, refer to “Headers (httpClient-HttpHeaders)”.
Config	additional properties	Contains additional configuration items. For property descriptions, refer to “Config (httpClient-WebsocketConfig)”.
Authenticator	additional properties	Configures the authentication method. For property descriptions, refer to “Authenticator (httpClient-Http Authenticator)”.

Property	Value	Description
Request Body	additional properties	Configures the request body content. For property descriptions, refer to "Request Body (httpRequestBody)".
Connected	read-only	Indicates if the component is connected to the client (`true`) or not (`false`.
Last Connected	read-only	Displays the last time the device connected to the server.
Last Sent Message	read-only	Displays the last message sent to the server.
Last Sent Time	read-only	Displays when the last message was sent to the server.
Last Received Message	read-only	Displays the last message received from the server.
Last Received Time	read-only	Displays when the last message was received by the server.

Actions

- **Connect** manually attempts a connection to the WebSocket.
- **Disconnect** removes the connection.
- **Send** attempts to connect to the WebSocket to deliver the message.

Chapter 6 Plugins

Topics covered in this chapter

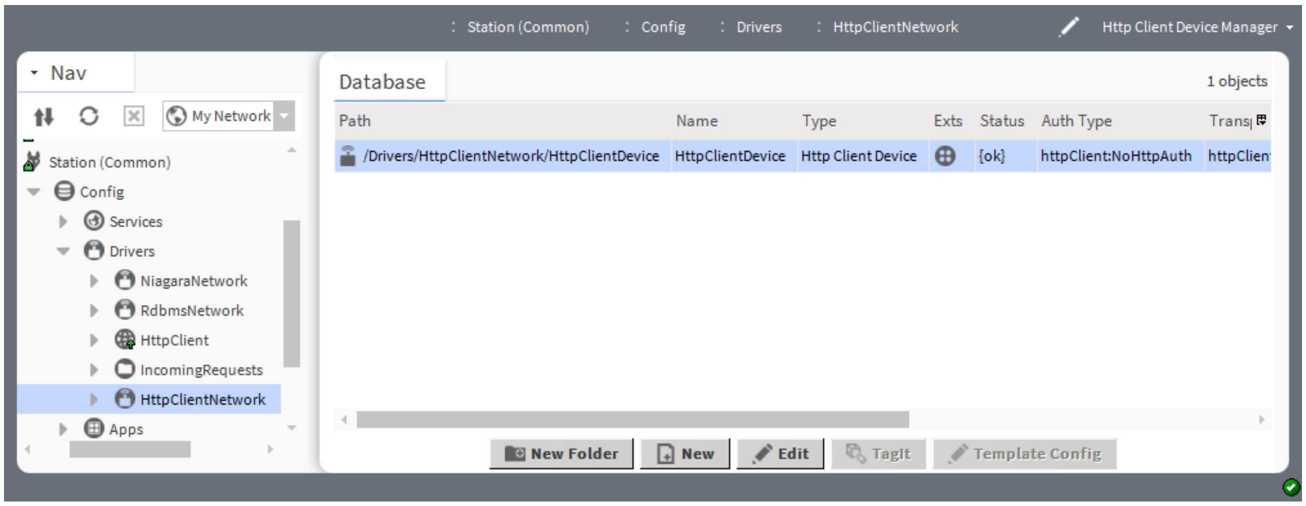
♦ Http Client Device Manager
♦ Http Client Point Manager

Plugins provide views of components and can be accessed in many ways. For example, double-click a component in the Nav tree to see its default view. In addition, you can right-click on a component and select from its Views menu.

Http Client Device Manager

This is the default view of **HttpClientNetwork**.

Figure 45 Http Client Device Manager



Path	Name	Type	Exts	Status	Auth Type	Trans
/Drivers/HttpClientNetwork/HttpClientDevice	HttpClientDevice	Http Client Device	⊕	{ok}	httpClient:NoHttpAuth	httpClien

To open this view, expand **Config→Drivers** and double-click the **HttpClientNetwork** component.

Columns

Column	Description
Path	Reports the location of the device.
Name	Reports the name of the device.
Type	Reports the type of device
Exts	⊕(Point Manager icon) opens the Http Client Point Manager view.
Status	Indicates the current state of the device.
Auth Type	Reports the authentication type of device.
Transport Type	Reports the transport type of device.
Host Address	Reports the IP address (URL) of the device.

Column	Description
Port	Identifies the HTTP port for the device.
Method	Reports the request method of the device.

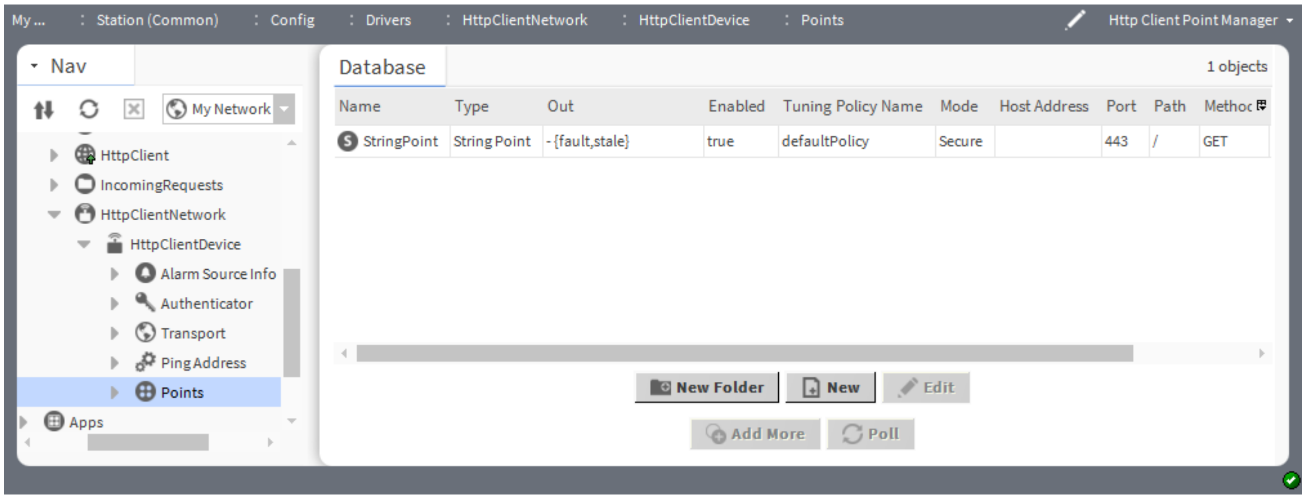
Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Template Config** accesses the station template that defines configuration options. You would select a template to set up the device with pre-configured properties.

Http Client Point Manager

This manager provides access to the proxy points mapped into the **PointDeviceExt** component.

Figure 46 Http Client Point Manager view



Name	Type	Out	Enabled	Tuning Policy Name	Mode	Host Address	Port	Path	Method
StringPoint	String Point	- {fault,stale}	true	defaultPolicy	Secure		443	/	GET

To open this view, expand **Config→Drivers→HttpClientNetwork→HttpClientDevice** and double-click the **Points**.

Columns

Column	Description
Name	Reports the name of the point.
Type	Reports the type of point
Facets	Reports the facets setting of the point.
Out	Represents the point slot that contains the value to output
Status	Indicates the current state of the device.
Enabled	Reports if the point is functional.
Tuning Policy Name	Displays the selected tuning policy name.
Mode	Displays the response mode.

Column	Description
Host Address	Reports the IP address (URL) of the device.
Port	Identifies the HTTP port for the device.
Path	Reports the URL to the point.
Method	Reports the request method of the device.
Source Type	Reports the source of the point.
Poll Frequency	Indicates how frequently the device is polling the data.

Buttons

- **New Folder** creates a new folder for devices. Each such folder provides its own set of manager views.
- **New** creates a new device record in the database.
- **Edit** opens the device's database record for updating.
- **Add More** allows to add more slots to the point.
- **Poll** allows to poll the device.

Chapter 7 Windows

Topics covered in this chapter

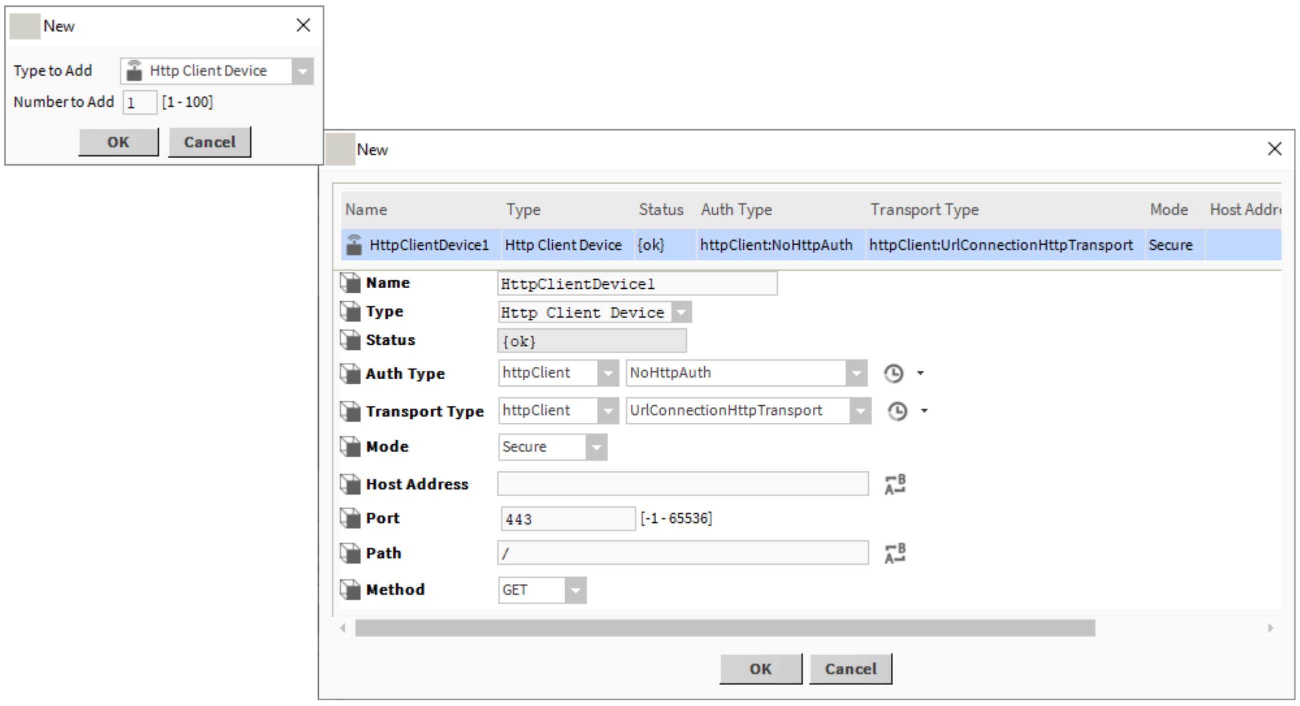
- ♦ New device windows
- ♦ Edit device window
- ♦ Add slot window
- ♦ Populate From Url window
- ♦ New point window
- ♦ Add More window

Windows create and edit database records or collect information when accessing a component. You access them by dragging a component from a palette into a station or by clicking a button.

Windows do not support **On View (F1)** and **Guide on Target** help. To learn about the information each contains, search the help system for key words.

New device windows

This window add device records. This topic documents only some of a device component's properties.



Type to Add	Http Client Device
Number to Add	1 [1 - 100]

Name	Type	Status	Auth Type	Transport Type	Mode	Host Addr
HttpClientDevice1	Http Client Device	{ok}	httpClient:NoHttpAuth	httpClient:UrlConnectionHttpTransport	Secure	

Name	HttpClientDevice1
Type	Http Client Device
Status	{ok}
Auth Type	httpClient NoHttpAuth
Transport Type	httpClient UrlConnectionHttpTransport
Mode	Secure
Host Address	
Port	443 [-1 - 65536]
Path	/
Method	GET

Figure 47 New device windows

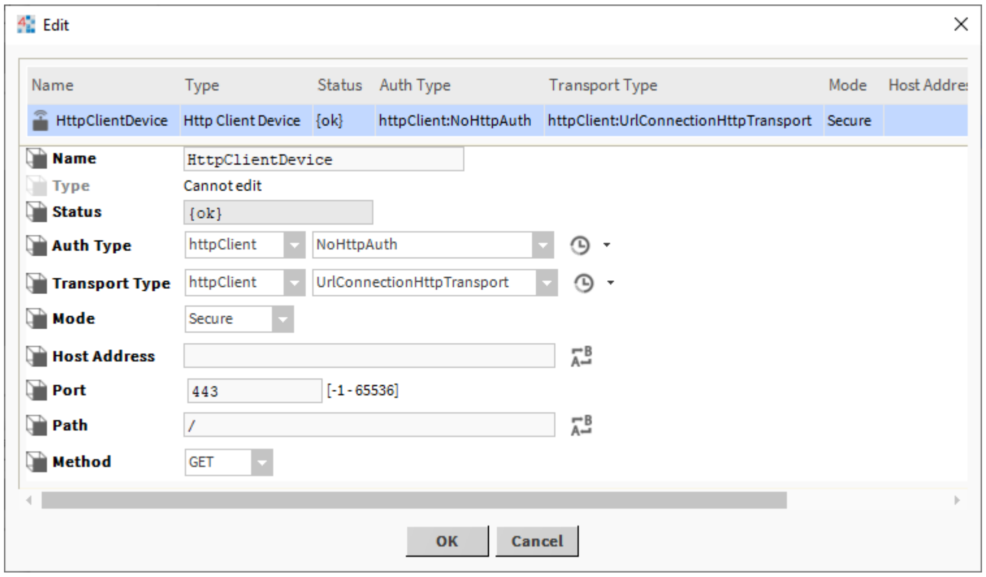
To open this window, expand **Config→Drivers** and double-click **HttpClientNetwork**. **Http Client Device Manager** opens, click the **New** button. Select the device from the drop-down list, and click **OK**. Another New window opens, where other parameters for the device are configured.

Property	Value	Description
Name	text	Provides descriptive text that reflects the identity of the entity or logical grouping.
Type	drop-down list	Specifies the type of device.
Status	read-only	Reports the current condition of the entity as of the last refresh: {alarm}, {disabled}, {down}, {fault}, {ok}, {stale}, {unackedAlarm}
Auth Type	drop-down list (defaults to httpClient)	Selects the type of user authentication from among these methods: • HTTP Basic • HTTP Digest • Niagara SCRAM-SHA • Bearer token • Cookies from a previous request Selecting **`Auth Type`** and saving updates the **`Config`** property below allowing further settings to be applied.
Transport Type	drop-down list (defaults to httpClient)	Switches the underlying transport layer between the standard JRE (`UrlConnectionHttpTransport`) and the third-party OKHttp library (`OKHttp Transport`).
Mode	drop-down list	Selects the security mode. `Secure`: Secure mode refers to https on port 443 by default. `Insecure`: Insecure mode means http without SSL and assumes port 80 by default.
Host Address	URL	Defines the URL for the client's address and parameters. This is the address to ping for a given device.
Port	Number (defaults 443)	Specifies the http port number.
Method	drop-down list (defaults to GET)	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.

Edit device window

This window edits the already added device records. This topic documents only some of a device component's properties.

Figure 48 Edit device window



Name	Type	Status	Auth Type	Transport Type	Mode	Host Addre
HttpClientDevice	Http Client Device	{ok}	httpClient:NoHttpAuth	httpClient:UrlConnectionHttpTransport	Secure	

Name: HttpClientDevice
Type: Cannot edit
Status: {ok}
Auth Type: httpClient NoHttpAuth
Transport Type: httpClient UrlConnectionHttpTransport
Mode: Secure
Host Address:
Port: 443 [-1 - 65536]
Path: /
Method: GET

To open this window, expand **Config→Drivers** and double-click **HttpClientNetwork**. **Http Client Device Manager** opens select the device which needs to be edited, Click **Edit**button. Select the device from the drop-down list, and click **OK**. Another New window opens, where other parameters for the device are configured.

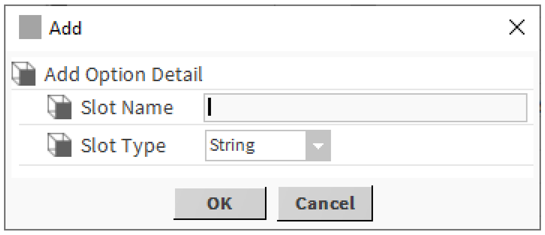
Property	**Value**	**Description**
Name	text	Provides descriptive text that reflects the identity of the entity or logical grouping.
Type	unavailable to edit	unavailable to edit
Status	read-only	Reports the current condition of the entity as of the last refresh: {alarm}, {disabled}, {down}, {fault}, {ok}, {stale}, {unackedAlarm}
Auth Type	drop-down lists (default to `httpClient`, `NoHttpAuth`)	Selects the type of user authentication from among these methods: • HTTP Basic • HTTP Digest • Niagara SCRAM-SHA • Bearer token • Cookies from a previous request Selecting **`Auth Type`** and saving updates the **`Config`** property below allowing further settings to be applied.
Transport Type	drop-down lists (default to `httpClient`, `UrlConnec-tionHttpTran-sport`)	Switches the underlying transport layer between the standard JRE (`UrlConnectionHttpTransport`) and the third-party OKHttp library (`OKHttp Transport`).
Mode	drop-down list	Selects the security mode.

Property	Value	Description
		`Secure`: Secure mode refers to https on port 443 by default. `Insecure`: Insecure mode means http without SSL and assumes port 80 by default.
Host Address	URL	Defines the URL for the client's address and parameters. This is the address to ping for a given device.
Port	Number (defaults 443)	Specifies the http port number.
Method	drop-down list	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.

Add slot window

This window adds a slot to the station.

Figure 49 Add slot window



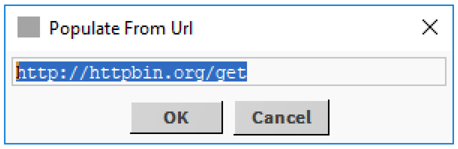
To open this window, expand **HttpClient**, right-click **Parameters** and click **Actions→Add**

Property	Value	Description
Slot Name	drop-down list	Identifies the slot. As soon as you type a letter, the available names appear.
Slot Type	drop-down list (defaults to `String`)	Selects the type of property to add. `String` defines a text string. `Boolean` defines a toggle. `Numeric` defines a numeric property.

Populate From Url window

This window fills in header values automatically based on a URL

Figure 50 Populate From Url window


Populate From Url

http://httpbin.org/get

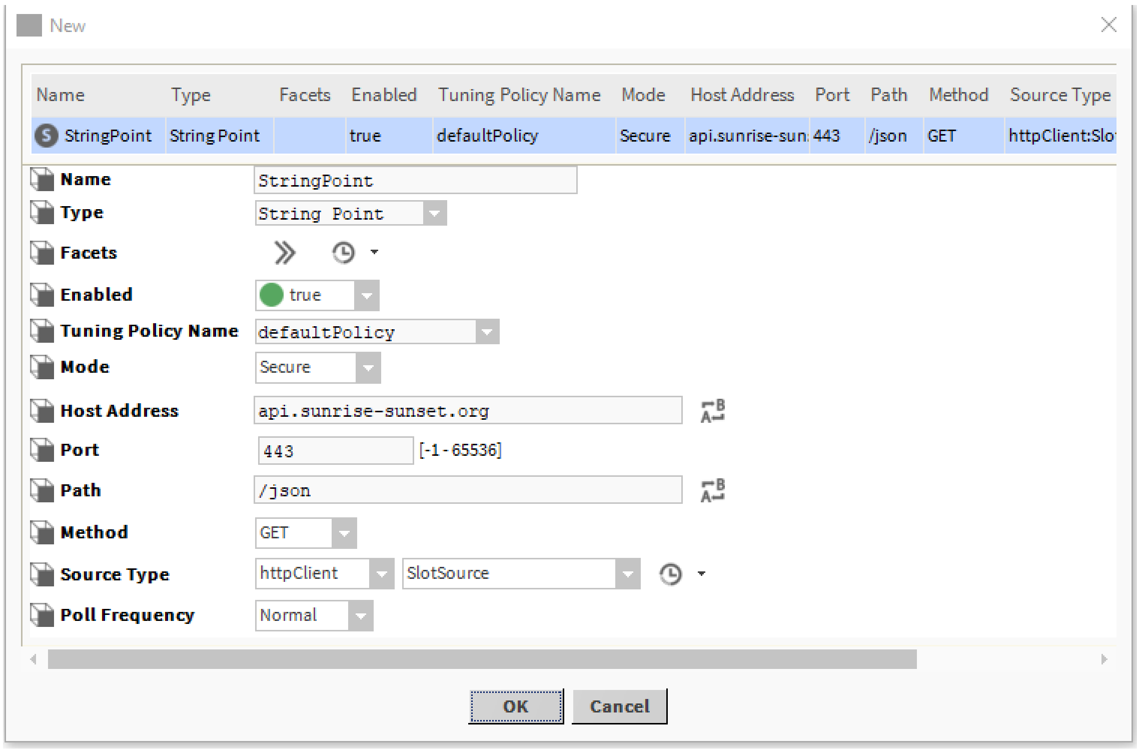
To open this window, right-click **Address** and click **Actions→Populate From Ul**.

Property	Value	Description
blank field	URL	Defines the request URL that contains the header values.

New point window

This window configures StringPoints.

Figure 51 New point window


New

Name	Type	Facets	Enabled	Tuning Policy Name	Mode	Host Address	Port	Path	Method	Source Type
StringPoint	String Point		true	defaultPolicy	Secure	api.sunrise-sun	443	/json	GET	httpClient:Slo

Name	StringPoint
Type	String Point
Facets	
Enabled	true
Tuning Policy Name	defaultPolicy
Mode	Secure
Host Address	api.sunrise-sunset.org
Port	443 [-1 - 65536]
Path	/json
Method	GET
Source Type	httpClient SlotSource
Poll Frequency	Normal

To open this window expand **Config→Drivers→HttpClientNetwork→HttpClientDevice**, double-click **Points** and click **New**.

In addition to the standard properties (Facets, Enabled and Tuning Policy Name), these properties configure an Http Client Driver's StringPoint.

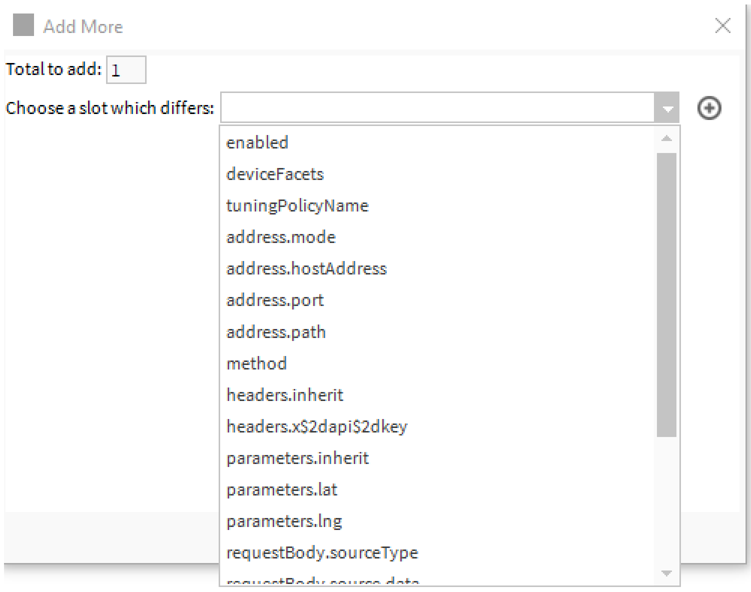
Property	Value	Description
Name	text	Provides descriptive text that reflects the identity of the entity or logical grouping.
Type	drop-down list	Specifies the type of device.
Mode	drop-down list	Selects the security mode.

Property	Value	Description
		`Secure`: Secure mode refers to https on port 443 by default. `Insecure`: Insecure mode means http without SSL and assumes port 80 by default.
Host Address	URL	Defines the URL for the client's address and parameters. This is the address to ping for a given device.
Port	number (defaults 443)	Specifies the http port number.
Path	text	Defines the path to the resource in the web service (that is, the path after the host address).
Method	drop-down list (defaults to GET)	Selects a request method from: `GET`: is used to request data from a specified resource. `POST`: is used to send data to a server to create/update a resource. The data sent to the server with POST is stored in the request body of the HTTP request. `PUT`: is used to send data to a server to create/update a resource. The difference between the POST and PUT request is that the PUT request are unchanged.
Source Type	drop-down lists	Identifies the source of the data. This is a point.
Poll Frequency	drop-down list (defaults to `Normal`)	Selects among three rates (Fast, Normal and Slow) to determine how often to query the component for its value. The network's Poll Service or Poll Scheduler defines these rates in hours, minutes and seconds. For example: `Fast` may set polling frequency to every second. `Normal` may set poll frequency to every five seconds. `Slow` may set poll frequency to every 30 seconds. This property applies to all proxy points.

Add More window

This window replicates configuration actions for multiple components.

Figure 52 Add More window



To open this window, expand the client, **Points** folder and point, right-click the **Proxy Ext** and click **Action-s→Add More**.

Property	Value	Description
Total to add	number	Selects how many components to add.
Choose a slot which differs:	drop-down list	Selects a slot from the source component to modify.

Index