OPENDCS

HydroJSON Implementation

Document Revision 3

July, 2022

This Document is part of the OpenDCS Software Suite for environmental data acquisition and processing. The project home is: <https://github.com/opendcs/opendcs>

See INTENT.md at the project home for information on licensing.

Table of Contents

1 Introduction 1

1.1 Document History 2

1.2 Glossary 3

2 Installation and Configuration 4

3 Cove Software Demo Installation 5

4 REST Method Reference 6

4.1 Authentication and Authorization 7

4.2 Check a Token 8

4.3 GET Reference List(s) 9

4.4 Network Lists 11

4.4.1 GET netlistrefs 11

4.4.2 GET netlist 11

4.4.3 POST netlist 12

4.4.4 DELETE netlist 12

4.5 DECODES Data Source Records 13

4.5.1 GET datasourcerefs 13

4.5.2 GET datasource 13

4.5.3 POST datasource 14

4.5.4 DELETE datasource 14

4.6 DECODES Site Records 15

4.6.1 GET siterefs 15

4.6.2 GET site 15

4.6.3 POST site 16

4.6.4 DELETE site 16

4.7 DECODES Platform Configurations 17

4.7.1 GET configrefs 17

4.7.2 GET config 18

4.7.3 POST config 21

4.7.4 DELETE config 21

4.8 DECODES Platform Records 22

4.8.1 GET platformrefs 22

4.8.2 GET platform 23

4.8.3 POST platform 24

4.8.4 DELETE platform 25

4.9 DECODES Presentation Group Records 26

4.9.1 GET presentationrefs 26

4.9.2 GET presentation 26

4.9.3 POST presentation 28

4.9.4 DELETE presentation 28

4.10 DECODES Routing Spec Records 29

4.10.1 GET routingrefs 29

4.10.2 GET routing 29

4.10.3 POST routing 30

4.10.4 DELETE routing 30

4.11 Schedule Entry Methods 31

4.11.1 GET schedulerefs 31

4.11.2 GET schedule 31

4.11.3 POST schedule 32

4.11.4 DELETE schedule 32

4.12 Loading Application Records 33

4.12.1 GET apprefs 33

4.12.2 GET app 34

4.12.3 POST app 34

4.12.4 DELETE app 34

4.13 Algorithm Methods 35

4.13.1 GET algorithmrefs 35

4.13.2 GET algorithm 35

4.13.3 POST algorithm 36

4.13.4 DELETE algorithm 36

4.14 Computation Methods 37

4.14.1 GET computationrefs 37

4.14.2 GET computation 37

4.14.3 POST computation 38

4.14.4 DELETE computation 39

4.15 TSDB Properties Methods 40

4.15.1 GET tsdb\_properties 40

4.15.2 POST tsdb\_properties 40

4.16 Engineering Unit Methods 41

4.16.1 GET unitlist 41

4.17 Data Type Methods 42

4.17.1 GET datatypelist 42

4.18 Retrieving Property Specs 43

4.19 Retrieve a Raw DCP Message 45

5 Time Series Methods 46

5.1 GET Time Series Catalog 47

5.2 GET Time Series Data 49

5.3 POST Convert Netlist from “.nl” File 52

# Introduction

HydroJSON is a JSON (Java Script Object Notation) standard for interchanging hydro, meteorological, and other environmental data. It was developed by the U. S. Army Corps of Engineers. The original defining page can be found here:

https://github.com/gunnarleffler/hydroJSON

OpenDCS implements HydroJSON as a Web Application running under Apache Tomcat. This document will define the calls that are implemented by OpenDCS and the returned data structures in detail.

The backend database for data delivered can be either CWMS (Corps Water Management System) or OpenTSDB (Open Time Series Database – a module within OpenDCS).

## Document History

Revision 1, August 2020 – Original document release.

## Glossary

JSON Java Script Object Notation

WAR Web Archive – deployable web applications are typically distributed as WAR files.

# Installation and Configuration

Configure for database

Build the WAR

Deploy to TOMCAT

Troubleshooting

# Cove Software Demo Installation

# REST Method Reference

The following methods are implemented for OpenTSDB HydroJSON. Following the table, subsections will provide details on each.

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Arguments | Section | Description |
| credentials | none | 4.1 | POST method to login to system and retrieve a new token. |
| check | token | 4.2 | GET method, passed a token value, returns complete token if valid. |
| reflists | name, token | 4.3 | GET method. Optional arguments are name of reference list desired, and a token. Returns reference list(s). |
| netlists | name, token | 4.4.2 | GET method. Optional arguments are name of network list desired, and a token. Returns network list(s). |
| netlist | token | 4.4 | POST method. Valid token required. POST data is a JSON Network List. |
| tscatalog | site, param, interval, active, token | 4.5 | Get method to return a catalog of Time Series Identifiers. All arguments are optional and can be used for filtering. |
| tsdata | site, param, interval, active, since, until, token | 5.2 | Get method to return Time Series Identifiers containing Time Series Data. All arguments optional and can be used for filtering. |
| platforms | tmtype, token | 4.8.1 | List platforms of a given transport medium type. |
| cnvtnl | token | 5.3 | Convert LRGS “.nl” file to JSON NetList |

The Root URL for the service, unless changed during deployment, is OHydroJson. Thus to access the “platforms” method on a tomcat server running on localhost on port 8080, the complete URL would be:

http://localhost:8080/OHydroJson/platforms

## Authentication and Authorization

Authentication and authorization is required for any method that modifies the database.

The ‘credentials’ POST method is used to obtain a new token. The POST data must be of the form:

{ "username" : "***UserNameForLogin***”, "password" : "***PasswordForLogin***" }

The user name and password provided must be a valid login for the underlying database. Also, that user must be assigned either of the roles OTSDB\_ADMIN or OTSDB\_MGR. If successful, a token is returned with the following JSON structure:

{"lastUsed":1596807452610,"token":"9fe6390676c7dca9","username":"xyz"}

* The “username” argument will be the same username passed to the POST method.
* The “token” argument is a randomized hexadecimal string that must be passed to subsequent methods that require authentication.
* The “lastUsed” argument is a long integer number of milliseconds since the epoch (Jan 1 1970 at midnight UTC).

Tokens are valid for a finite period of time. Every time a token is used (i.e. passed to a method), its lastUsed field is updated. If more than 3 hours goes by without a token being used, it is removed from the server and is no longer valid.

If the passed username or password is not valid, or if the user is not assigned the required role, an HTTP 401 error is returned with an explanatory message. It will typically be one of the following:

FATAL: password authentication failed for user “xyz”

User xyz does not have OTSDB\_ADMIN or OTSDB\_MGR privilege – Not Authorized.

## Check a Token

The ‘check’ GET method can be called with a token argument. Example:

http://localhost:8080/OHydroJson/check?token=6d34fa0e3bb72fcd

If the token is valid, the token JSON object will be returned in the same format described above for authentication.

If the token is not valid, HTTP 410 is returned. The body of the response is JSON in the form:

{"errMessage":"Token '6d34fa0e3bb72fcd' does not exist.","status":410}

## GET Reference List(s)

The ‘reflists’ GET method will return all reference lists or a specific reference list.

Example:

http://localhost:8080/OHydroJson/reflists?name=TransportMediumType

Authentication is not required for this method, but if a token argument is provided the lastUser timer in the token will be updated.

If no “name” argument is provided, then all reference lists in the database are returned.

The JSON structure of the returned data looks like this:

{

"TransportMediumType": {

"enumName": "TransportMediumType",

"items": {

"iridium": {

"description": "Iridium IMEI",

"value": "iridium"

},

"other": {

"description": "Other",

"value": "other"

},

"incoming-tcp": {

"description": "Incoming TCP on a Listening Socket",

"value": "incoming-tcp"

},

"goes": {

"description": "GOES DCP",

"value": "goes"

},

"goes-self-timed": {

"description": "GOES DCP Self-Timed Message",

"value": "goes-self-timed"

},

"polled-modem": {

"description": "Polled via modem",

"value": "polled-modem"

},

"polled-tcp": {

"description": "Polled via TCP Socket",

"value": "polled-tcp"

},

"data-logger": {

"description": "Electronic Data Logger File",

"value": "data-logger"

}

}

}

}

The following reference lists are currently available:

* DataSourceType – Used in Database Editor to link a data source name to a Java class
* PortType – Used in the polling interface
* ScriptType – A list of DECODES Platform Configuration Script types (reserved for future use
* StatisticsCode – Valid statistics codes that can be used in a time series identifier
* SiteNameType – Known site name types, e.g. NWSHB5 (National Weather Service Handbook 5), USGS (Site Number), CODWR (Colorada Dept of Water Resources).
* DataConsumer – Links a data consumer type name to Java Code (e.g. File, Directory, Socket)
* Measures – A list of physical attributes measured by an engineering unit (e.g. force, temperature, mass)
* UnitConversionAlgorithm – A list of algorithms for unit conversion
* DataOrder – Ascending (oldest first) or Descending (newest first) time order
* GroupType – Used for annotation in time series groups
* LoggerType – Used in the polling interface
* LookupAlgorithm
* EquationScope
* UnitFamily – Metric, English, or Universal
* OutputFormat – Links a name to Java code for formatting output data (e.g. SHEF, CSV)
* DataTypeStandard – Known standards for specifying data type (e.g. SHEF-PE, CWMS, HDB)
* TransportMediumType – A Transport Medium uniquely identifies a platform. There are several types: GOES ID, Iridium IMEI, Polled Identifier, etc.
* Season – User can create any number of seasons that start/end at specified time of year
* EquipmentType
* ApplicationType – Computation Process, Comp Depends Daemon, DECODES Routing Scheduler, etc.
* RecordingMode

## Network Lists

A Network List is simply a list of Platforms.

### GET netlistrefs

The GET netlistrefs method returns references to network lists. It is intended to populate a pick list of network lists and does not contain all of the list elements.

Authentication is not required for this method, but if a token argument is provided the

Examples:

http://localhost:8080/OHydroJson/netlistrefs

http://localhost:8080/OHydroJson/netlistrefs?tmtype=goes

With no arguments, a list of network lists in the database is returned. The format is as follows:

[

{

"lastModifyTime": "2020-08-22T14:36:55.705Z[UTC]",

"name": "BFD-BMD",

"netlistId": 1,

"numPlatforms": 3,

"siteNameTypePref": "nwshb5",

"transportMediumType": "goes"

},

{

"lastModifyTime": "2020-12-15T17:51:04.194Z[UTC]",

"name": "goes2",

"netlistId": 6,

"numPlatforms": 3,

"siteNameTypePref": "nwshb5",

"transportMediumType": "goes"

}

]

Note that the list contents (i.e. the references to the platforms in the list) is not included. Rather a count of platforms in the list is given.

The “netlistId” element is a unique key to be used for retrieving entire lists.

### GET netlist

The ‘netlists’ GET method will return a specific network list in its entirety.

Example:

http://localhost:8080/OHydroJson/netlists?netlistid=1

Authentication is not required for this method, but if a token argument is provided the lastUser timer in the token will be updated.

The JSON structure of the returned data looks like this:

{

"items": {

"14159500": {

"description": "",

"platformName": "CGRO",

"transportId": "14159500"

},

"14372300": {

"description": "",

"platformName": "AGNO",

"transportId": "14372300"

}

},

"lastModifyTime": "2020-10-19T18:14:14.788Z[UTC]",

"name": "USGS-Sites",

"netlistId": 4,

"siteNameTypePref": "nwshb5",

"transportMediumType": "other"

}

Possible error codes:

* 406 (Not Acceptable): The required ‘netlistid’ parameter was missing in the URL.
* 404 (Not Found): The requested netlistid was not found in the database.

### POST netlist

The ‘netlist’ POST method requires a valid token. It takes a single network list in JSON format, as described for the GET method.

For creating a new network list, leave netlistId out of the passed data structure.

For overwriting an existing one, include the netlistId that was previously returned. The network list in the database is replaced with the one sent.

### DELETE netlist

The DELETE netlist method requires a valid token.

Required argument netlistid must be passed.

Error 405 will be returned if network list is used by one or more routing specs and cannot be deleted. The body of the error will be a message containing the name of the routing specs using the referenced netlist.

## DECODES Data Source Records

### GET datasourcerefs

Example:

http://localhost:8080/OHydroJson/datasourcerefs

This method returns a JSON list of DECODES Data Source records suitable for displaying in a table or pick-list. The returned structure contains only the high-level descriptive information about each data source. The returned structure is as follows:

{

"CDADATA-As-MBHydro": {

"arguments": "host=cdadata.wcda.noaa.gov, password=\*\*\*\*, username=mbhydro",

"dataSourceId": 5,

"name": "CDADATA-As-MBHydro",

"type": "lrgs",

"usedBy": 0

},

"edl-tcp-poll": {

"arguments": "availablePorts=20, maxBacklogHours=48, pollNumTries=3, portType=tcpclient, saveSessionFile=$DCSTOOL\_USERDIR/edl-polled/$SITENAME-$DATE",

"dataSourceId": 9,

"name": "edl-tcp-poll",

"type": "polled",

"usedBy": 1

},

"testgroup": {

"dataSourceId": 10,

"name": "testgroup",

"type": "hotbackupgroup",

"usedBy": 0

}

}

The arguments (properties) are represented by a string with comma delimiter. Passwords within the string are replaced with four asterisks.

### GET datasource

The GET datasource method returns a single DECODES data source with all of its detail. The integer argument datasourceid is required.

Example:

http://localhost:8080/OHydroJson/datasource?datasourceid=10

Returned structure:

{

"dataSourceId": 10,

"groupMembers": [

{

"dataSourceId": 4,

"dataSourceName": "Cove-LRGS"

},

{

"dataSourceId": 5,

"dataSourceName": "CDADATA-As-MBHydro"

},

{

"dataSourceId": 7,

"dataSourceName": "USGS-Web"

}

],

"name": "testgroup",

"props": {

"whatevs": "something 567",

"abc": "def"

},

"type": "hotbackupgroup",

"usedBy": 0

}

### POST datasource

The ‘datasource’ POST method requires a valid token. It takes a single datasource in JSON format, as described for the GET method.

For creating a new network list, leave datasourceId out of the passed data structure.

For overwriting an existing one, include the datasourceId that was previously returned by GET. The network list in the database is replaced with the one sent.

### DELETE datasource

The DELETE datasource method requires a valid token.

Required argument datasource id must be passed.

Error 405 will be returned if datasource list is used by one or more routing specs and cannot be deleted. The body of the error will be a message containing the name of the routing specs using the referenced netlist.

## DECODES Site Records

### GET siterefs

Example:

http://localhost:8080/OHydroJson/siterefs

This method returns a JSON list of DECODES Site records suitable for displaying in a table or pick-list. The returned structure contains only the numeric ID (unique), description, and an array of site names in the following format:

[

{

"description": "Iowa River at Oakville, IA (USGS)",

"siteId": 1,

"sitenames": {

"CWMS": "OKVI4",

"nwshb5": "OKVI4",

"usgs": "05465700",

"local": "OKVI4"

}

},

{

"description": "IOWA RIVER NEAR ROWAN 4NW",

"siteId": 2,

"sitenames": {

"CWMS": "ROWI4",

"USGS": "05449500"

}

},

{

"description": "Site used for Regression Testing",

"siteId": 3,

"sitenames": {

"CWMS": "TESTSITE1"

}

}

]

The numeric ID may be used in subsequent calls to the “site” method.

### GET site

Example:

http://localhost:8080/OHydroJson/site?siteid=3

This method returns a JSON representation of a single, complete DECODES Site record. The structure is as follows:

{

"country": "USA",

"description": "Barre Falls Dam, Ware River",

"elevUnits": "M",

"elevation": 234.7,

"latitude": "42.4278",

"longitude": "-72.0261",

"nearestCity": "Barre Falls Dam",

"properties": {

"some": "thing",

"something": "else"

},

"region": "",

"siteId": 7,

"sitenames": {

"CWMS": "BFD",

"NWSHB5": "BFD"

},

"state": "MA",

"timezone": "America/New\_York"

}

### POST site

The POST site method requires a valid token. It takes a single DECODES site record in JSON format, as described above for GET site.

For creating a new site, leave siteId out of the passed data structure.

For overwriting an existing one, include the siteId that was previously returned. The site list in the database is replaced with the one sent.

### DELETE site

The DELETE site method requires a valid token.

Required argument siteid must be passed.

Error 405 will be returned if site is used by one or more platforms and cannot be deleted.

## DECODES Platform Configurations

### GET configrefs

Example:

http://localhost:8080/OHydroJson/configrefs

This method returns a JSON list of DECODES Config records suitable for displaying in a table or pick-list. The returned structure contains an array in the following format:

[

{

"configId": 1,

"description": "WSC SHEF - 2 sensors - HG, VB",

"name": "Shef-WSC-Hydro-RCOYCHER",

"numPlatforms": 1

},

{

"configId": 2,

"description": "WSC SHEF - 2 sensors - QR=QF, HG",

"name": "Shef-WSC-Hydro-RBRDDDVH",

"numPlatforms": 1

},

{

"configId": 3,

"description": "WSC SHEF - 2 sensors - HG, VB",

"name": "Shef-WSC-Hydro-RBLOCLEE",

"numPlatforms": 1

},

{

"configId": 4,

"description": "AE SHEF - 4 sensors - PC, TA, SW, YB=VB",

"name": "Shef-AE-Met-SESK",

"numPlatforms": 1

},

{

"configId": 5,

"description": "WSC SHEF - 2 sensors - HG, VB",

"name": "Shef-WSC-Hydro-RBULLRES",

"numPlatforms": 1

},

{

"configId": 6,

"description": "WSC SHEF - 2 sensors - HG, VB",

"name": "Shef-WSC-Hydro-RREDBIN",

"numPlatforms": 1

}

]

The numeric ID may be used in subsequent calls to the “config” method.

### GET config

Example:

http://localhost:8080/OHydroJson/config?configid=12

This method returns a JSON representation of a single, complete DECODES Config record. Example:

{

"configId": 12,

"configSensors": [

{

"dataTypes": {

"NL-SHEF": "WL",

"SHEF-PE": "HG",

"W-SHEF": "WL"

},

"properties": {},

"recordingInterval": 3600,

"recordingMode": "F",

"sensorName": "WL",

"sensorNumber": 1,

"timeOfFirstSample": 0

},

{

"dataTypes": {

"SHEF-PE": "VB"

},

"properties": {},

"recordingInterval": 3600,

"recordingMode": "F",

"sensorName": "VB",

"sensorNumber": 4,

"timeOfFirstSample": 0

}

],

"description": "WSC SHEF - 2 sensors - HG, VB",

"name": "Shef-WSC-Hydro-RVERMMAR",

"numPlatforms": 0,

"scripts": [

{

"dataOrder": "D",

"formatStatements": [

{

"format": "s(50,':',DONE),x,F(F,A,10d' ')",

"label": "getlabel",

"sequenceNum": 0

},

{

"format": "s(12,'#',getlabel),x,f(mint,a,3d' +-',1),32(w,c(N,skiphg),F(S,A,12d' +-:',1)), >GETLABEL",

"label": "hg",

"sequenceNum": 1

},

{

"format": "s(12,'.',getlabel),-2x,32(w,c(N,getlabel),F(S,A,12d' +-:',4)), >GETLABEL",

"label": "vb",

"sequenceNum": 2

},

{

"format": "s(12,'.',enddata),-2x,32(w,c(N,enddata),F(S,A,12d' +-:',4)), >GETLABEL",

"label": "x-nointerval",

"sequenceNum": 3

},

{

"format": ">vb",

"label": "battery",

"sequenceNum": 4

},

{

"format": "-3x,>getlabel",

"label": "enddata",

"sequenceNum": 5

},

{

"format": "/",

"label": "error",

"sequenceNum": 6

},

{

"format": "/,>getlabel",

"label": "done",

"sequenceNum": 7

},

{

"format": "-2x,5(S(50,':',done),S(3,'HG',enddata))>getlabel",

"label": "skiphg",

"sequenceNum": 8

},

{

"format": "s(12,'#',getlabel),x,f(mint,a,3d' +-',6),32(w,c(N,enddata),F(S,A,12d' +-:',6)), >GETLABEL",

"label": "pr",

"sequenceNum": 9

},

{

"format": ">pr",

"label": "pc",

"sequenceNum": 10

},

{

"format": ">hg",

"label": "hk",

"sequenceNum": 11

},

{

"format": "s(12,'#',getlabel),x,f(mint,a,3d' +-',7),32(w,c(N,enddata),F(S,A,12d' +-:',7)), >GETLABEL",

"label": "tx",

"sequenceNum": 12

},

{

"format": "s(12,'#',getlabel),x,f(mint,a,3d' +-',8),32(w,c(N,enddata),F(S,A,12d' +-:',8)), >GETLABEL",

"label": "tn",

"sequenceNum": 13

},

{

"format": "s(12,'.',enddata),-2x,32(w,c(N,enddata),F(S,A,12d' +-:',4)), >GETLABEL",

"label": "vb-x",

"sequenceNum": 14

},

{

"format": "s(12,'#',getlabel),x,f(mint,a,3d' +-',9),32(w,c(N,skiphg),F(S,A,12d' +-:',9)), >GETLABEL",

"label": "hh",

"sequenceNum": 15

}

],

"name": "ST",

"scriptSensors": [

{

"sensorNumber": 1,

"unitConverter": {

"a": 0,

"algorithm": "none",

"b": 0,

"c": 0,

"d": 0,

"e": 0,

"f": 0,

"fromAbbr": "raw",

"toAbbr": "M"

}

},

{

"sensorNumber": 4,

"unitConverter": {

"a": 0,

"algorithm": "none",

"b": 0,

"c": 0,

"d": 0,

"e": 0,

"f": 0,

"fromAbbr": "raw",

"toAbbr": "V"

}

}

]

}

]

}

### POST config

The POST config method requires a valid token. It takes a single DECODES Platform Configuration record in JSON format, as described above for GET config.

For creating a new config, leave configId out of the passed data structure.

For overwriting an existing one, include the configId that was previously returned. The configuration in the database is replaced with the one sent.

### DELETE config

The DELETE config method requires a valid token.

Required argument configid must be passed.

Error 405 will be returned if the referenced configuration is used by one or more platforms and cannot be deleted.

## DECODES Platform Records

### GET platformrefs

The GET platformrefs method returns a list of platforms, optionally with a given Transport Medium type. It accepts the following arguments:

* tmtype – the transport medium type desired. If not provided, all types are returned. The method will return any platform that has a transport medium with the given type. The returned data structure will still contain all of the transport media in the platform, but the list of platforms will be filtered to only include platforms with a TM of the given type:
  + NOTE: medium type “goes” will match either goes-self-timed or goes-random.
* token: Not required, but if included, the token lastUsed time will be updated.

Data Structure TBD but will include

* name – an index into the hashed set. Combination of site name and designator
* agency – The agency that owns and/or maintains this platform
* configId – Numeric surrogate key to the configuration record
* description
* platformId - Numeric surrogate key to the platform record
* siteId - Numeric surrogate key to the site record
* transportMedia – a list of tmtype/tm id pairs.

The returned JSON data is structured as follows:

{

"BFD": {

"agency": "CWMS",

"configId": 5,

"description": "Barre Falls Dam, Ware River, MA",

"name": "BFD",

"platformId": 4,

"siteId": 7,

"transportMedia": {

"goes-random": "CE3234CC",

"goes-self-timed": "CE3234CC"

}

},

"BMD-tailwater": {

"agency": "CWMS",

"configId": 7,

"description": "Ball Mountain TW",

"designator": "tailwater",

"name": "BMD-tailwater",

"platformId": 6,

"siteId": 8,

"transportMedia": {

"goes-random": "CE3941F4",

"goes-self-timed": "CE3941F4"

}

},

"BMD": {

"agency": "CWMS",

"configId": 6,

"description": "Ball Mountain Dam, West River,VT",

"name": "BMD",

"platformId": 5,

"siteId": 8,

"transportMedia": {

"goes-random": "CE31D030",

"goes": "CE31D030",

"goes-self-timed": "CE31D030"

}

}

}

### GET platform

Example:

http://localhost:8080/OHydroJson/platform?platformid=5

This method returns a JSON representation of a single, complete DECODES Platform record. The following structure is returned:

{

"agency": "CWMS",

"configId": 6,

"description": "Ball Mountain Dam, West River,VT",

"lastModified": "2022-01-21T14:18:21.176Z[UTC]",

"platformId": 5,

"platformSensors": [

{

"max": 120,

"min": -40,

"sensorNum": 1,

"sensorProps": {}

},

{

"sensorNum": 1,

"sensorProps": {}

},

{

"sensorNum": 1,

"sensorProps": {}

},

{

"sensorNum": 1,

"sensorProps": {}

},

{

"actualSiteId": 5,

"sensorNum": 1,

"sensorProps": {}

}

],

"production": false,

"properties": {},

"siteId": 8,

"transportMedia": [

{

"assignedTime": 2095,

"baud": 0,

"channelNum": 161,

"dataBits": 0,

"doLogin": false,

"mediumId": "CE31D030",

"mediumType": "goes-self-timed",

"parity": "U",

"scriptName": "ST",

"stopBits": 0,

"timeAdjustment": 0,

"timezone": "UTC",

"transportInterval": 3600,

"transportWindow": 5

},

{

"baud": 0,

"channelNum": 129,

"dataBits": 0,

"doLogin": false,

"mediumId": "CE31D030",

"mediumType": "goes-random",

"parity": "U",

"scriptName": "RD",

"stopBits": 0,

"timeAdjustment": 0,

"timezone": "UTC",

"transportInterval": 900

},

{

"baud": 0,

"channelNum": 235,

"dataBits": 0,

"doLogin": false,

"mediumId": "CE31D030",

"mediumType": "goes",

"parity": "U",

"scriptName": "Network\_Tower",

"stopBits": 0,

"timeAdjustment": 0,

"timezone": "UTC",

"transportInterval": 900

}

]

}

### POST platform

The POST platform method requires a valid token. It takes a single DECODES Platform record in JSON format, as described above for GET.

For creating a new platform, leave platformId out of the passed data structure.

For overwriting an existing one, include the platformId that was previously returned. The platform in the database is replaced with the one sent.

### DELETE platform

The DELETE platform method requires a valid token.

Required argument platformid must be passed.

## DECODES Presentation Group Records

### GET presentationrefs

Returns a list of references to presentation groups suitable for displaying a list. The returned structure is:

[

{

"groupId": 1,

"lastModified": "2022-01-14T14:43:47.297Z[UTC]",

"name": "CWMS-English",

"production": false

},

{

"groupId": 2,

"lastModified": "2020-05-11T20:23:20.103Z[UTC]",

"name": "CWMS-Metric",

"production": false

},

{

"groupId": 3,

"lastModified": "2020-05-11T20:23:20.118Z[UTC]",

"name": "SHEF-English",

"production": false

},

{

"groupId": 4,

"lastModified": "2020-05-11T20:33:10.275Z[UTC]",

"name": "regtest",

"production": false

},

{

"groupId": 5,

"lastModified": "2022-03-04T18:17:58.885Z[UTC]",

"name": "NRCS",

"production": true

},

{

"groupId": 6,

"inheritsFrom": "CWMS-English",

"lastModified": "2021-03-05T14:06:18.435Z[UTC]",

"name": "hydrodcs",

"production": false

}

]

### GET presentation

Example:

http://localhost:8080/OHydroJson/presentation?groupid=4

This method returns a JSON representation of a single, complete DECODES Presentation Group record. The following structure is returned. Note that the optional min and max elements are not always present.

{

"elements": [

{

"dataTypeCode": "Elev",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"units": "ft"

},

{

"dataTypeCode": "Elev-Pool",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"units": "ft"

},

{

"dataTypeCode": "FLOW-HOLDOUT",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"units": "cfs"

},

{

"dataTypeCode": "FLOW-INFLOW",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"units": "cfs"

},

{

"dataTypeCode": "Temp",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"max": 212,

"min": 0,

"units": "degF"

},

{

"dataTypeCode": "Flow",

"dataTypeStd": "CWMS",

"fractionalDigits": 2,

"units": "cfs"

}

],

"groupId": 4,

"lastModified": "2022-03-04T19:00:21.752Z[UTC]",

"name": "regtest",

"production": false

}

### POST presentation

The POST presentation method requires a valid token. It takes a single DECODES Presentation Group in JSON format, as described above for GET.

For creating a new record, leave groupId out of the passed data structure.

For overwriting an existing one, include the groupId that was previously returned. The presentation group in the database is replaced with the one sent.

### DELETE presentation

The DELETE platform method requires a valid token.

Required argument groupid must be passed in the URL.

## DECODES Routing Spec Records

### GET routingrefs

Example: <http://localhost:8080/OHydroJson/routingrefs>

The returned structure is:

[

{

"dataSourceName": "OKVI4",

"destination": "pipe()",

"lastModified": "2020-05-11T20:24:53.052Z[UTC]",

"name": "OKVI4-input",

"routingId": 1

},

{

"dataSourceName": "OKVI4",

"destination": "opentsdb()",

"lastModified": "2022-01-14T14:50:50.982Z[UTC]",

"name": "OKVI4-ingest",

"routingId": 2

},

{

"dataSourceName": "MROI4-ROWI4",

"destination": "opentsdb()",

"lastModified": "2020-05-11T20:33:10.274Z[UTC]",

"name": "rs-MROI4-ROWI4",

"routingId": 3

},

{

"dataSourceName": "BFD-BMD",

"destination": "pipe()",

"lastModified": "2022-01-14T14:44:23.26Z[UTC]",

"name": "BFD-BMD-Decode-Test",

"routingId": 4

}

]

### GET routing

Example:

http://localhost:8080/OHydroJson/routing?routingid=20

This method returns a JSON representation of a single routing spec. The returned structure is:

{

"applyTimeTo": "Both",

"ascendingTime": true,

"dataSourceId": 4,

"dataSourceName": "Cove-LRGS",

"destinationArg": "some-directory-path",

"destinationType": "directory",

"enableEquations": true,

"goesChannels": [

123

],

"goesRandom": true,

"goesSelfTimed": true,

"goesSpacecraftCheck": true,

"goesSpacecraftSelection": "East",

"iridium": true,

"lastModified": "2022-03-22T17:44:15.941Z[UTC]",

"name": "test",

"netlistNames": [

"goes1"

],

"networkDCP": true,

"outputFormat": "emit-ascii",

"outputTZ": "EST5EDT",

"parityCheck": true,

"paritySelection": "Good",

"platformIds": [

"12345678",

"004085C8"

],

"platformNames": [

"MROI4"

],

"presGroupName": "CWMS-English",

"production": true,

"properties": {},

"qualityNotifications": true,

"routingId": 20,

"settlingTimeDelay": true,

"since": "2022/065 00:00:00"

}

### POST routing

The POST routing method requires a valid token. It takes a single DECODES Routing Spec in JSON format, as described above for GET.

For creating a new record, leave routingId out of the passed data structure.

For overwriting an existing one, include the routingId that was previously returned. The routing spec in the database is replaced with the one sent.

### DELETE routing

The DELETE routing method requires a valid token.

Required argument routingid must be passed in the URL.

## Schedule Entry Methods

### GET schedulerefs

Example: <http://localhost:8080/OHydroJson/schedulerefs>

The returned structure is:

[

{

"appName": "RoutingScheduler",

"enabled": false,

"lastModified": "2020-12-15T17:52:13.934Z[UTC]",

"name": "goes1",

"routingSpecName": "goes1",

"schedEntryId": 9

},

{

"appName": "RoutingScheduler",

"enabled": false,

"lastModified": "2020-12-15T17:53:06.043Z[UTC]",

"name": "goes2",

"routingSpecName": "goes2",

"schedEntryId": 10

},

{

"enabled": false,

"lastModified": "2022-03-23T13:54:09.188Z[UTC]",

"name": "no\_app\_assigned",

"routingSpecName": "polltest",

"schedEntryId": 17

},

{

"appName": "RoutingScheduler",

"enabled": false,

"lastModified": "2022-01-14T14:45:01.336Z[UTC]",

"name": "junk",

"routingSpecName": "polltest",

"schedEntryId": 14

}

]

Note in the third entry above that appName may be omitted. In the database, a schedule entry may not (yet) be assigned to an application.

### GET schedule

Example:

http://localhost:8080/OHydroJson/schedule?scheduleid=21

This method returns a JSON representation of a single schedule entry. The returned structure is:

{

"appId": 14,

"appName": "RoutingScheduler",

"enabled": false,

"lastModified": "2022-03-28T18:02:13.659Z[UTC]",

"name": "something-else",

"routingSpecId": 9,

"routingSpecName": "goes1",

"schedEntryId": 21,

"timeZone": "America/New\_York"

}

### POST schedule

The POST schedule method requires a valid token. It takes a single DECODES Schedule Entry in JSON format, as described above for GET.

For creating a new record, leave schedEntryId out of the passed data structure.

For overwriting an existing one, include the schedEntryId that was previously returned. The routing spec in the database is replaced with the one sent.

### DELETE schedule

The DELETE schedule method requires a valid token.

Required argument scheduleid must be passed in the URL.

## Loading Application Records

### GET apprefs

Example: <http://localhost:8080/OHydroJson/apprefs>

The returned structure is:

[

{

"appId": 1,

"appName": "decodes",

"comment": "Manual DECODES Routing Specs",

"lastModified": "2020-05-11T20:23:21Z[UTC]"

},

{

"appId": 3,

"appName": "compedit",

"comment": "Computation Editing Application",

"lastModified": "2020-05-11T20:23:21Z[UTC]"

},

{

"appId": 4,

"appName": "compproc",

"appType": "computationprocess",

"comment": "Main Computation Process",

"lastModified": "2022-03-30T20:49:44Z[UTC]"

},

{

"appId": 8,

"appName": "utility",

"appType": "utility",

"comment": "Utility Program -- used by several apps",

"lastModified": "2022-01-14T15:28:40Z[UTC]"

},

{

"appId": 10,

"appName": "dcpmon",

"comment": "DCP Monitor Web Application",

"lastModified": "2020-05-11T20:23:21Z[UTC]"

},

{

"appId": 14,

"appName": "RoutingScheduler",

"appType": "routingscheduler",

"comment": "Schedules and Executes DECODES Routing Specs",

"lastModified": "2020-05-11T20:23:21Z[UTC]"

},

{

"appId": 15,

"appName": "compdepends",

"appType": "compdepends",

"comment": "Daemon to determine computation dependencies",

"lastModified": "2020-05-11T20:23:21Z[UTC]"

}

]

### GET app

Example:

<http://localhost:8080/OHydroJson/app?appid=4>

Return data format:

{

"appId": 4,

"appName": "compproc",

"appType": "computationprocess",

"comment": "Main Computation Process",

"lastModified": "2022-03-30T20:49:44Z[UTC]",

"manualEditingApp": false,

"properties": {

"fromName": "idk"

}

}

Note: appType may be omitted if it is not defined in the database.

### POST app

The POST app method requires a valid token. It takes a single DECODES Loading Application in JSON format, as described above for GET.

For creating a new record, leave appId out of the passed data structure.

For overwriting an existing one, include the appId that was previously returned. The app in the database is replaced with the one sent.

### DELETE app

The DELETE schedule method requires a valid token.

Required argument appid must be passed in the URL.

This operation will fail if the loading application is currently being used by any computations or schedule entries, or if it is currently running and has an active CP\_COMP\_PROC\_LOCK record.

## Algorithm Methods

### GET algorithmrefs

Example: <http://localhost:8080/OHydroJson/algorithmrefs>

The returned structure is:

[

{

"algorithmId": 1,

"algorithmName": "AlarmScreening",

"description": "Perform value, rate-of-change, stuck-sensor, and missing-val",

"execClass": "decodes.tsdb.alarm.AlarmScreeningAlgorithm",

"numCompsUsing": 2

},

{

"algorithmId": 2,

"algorithmName": "Bridge Clearance",

"description": "Computes bridge clearance by subtracting waterlevel from con",

"execClass": "decodes.tsdb.algo.BridgeClearance",

"numCompsUsing": 0

},

{

"algorithmId": 3,

"algorithmName": "CentralRunningAverageAlgorithm",

"description": "CentralRunningAverageAlgorithm averages single 'input' param",

"execClass": "decodes.tsdb.algo.CentralRunningAverageAlgorithm",

"numCompsUsing": 0

},

{

"algorithmId": 8,

"algorithmName": "ExpressionParserAlgorithm",

"description": "Allow up to 5 inputs labeled in1",

"execClass": "decodes.tsdb.algo.ExpressionParserAlgorithm",

"numCompsUsing": 0

},

…

]

### GET algorithm

Example:

<http://localhost:8080/OHydroJson/algorithm?algorithmid=4>

Return data format:

{

"algoScripts": [],

"algorithmId": 4,

"description": "Given two inputs, output the best one: If only one is present at the time-slice, output it. If one is outside the specified upper or lower limit (see properties) output the other. If both are acceptable, output the first one. Useful in situations where you have redundant sensors.",

"execClass": "decodes.tsdb.algo.ChooseOne",

"name": "ChooseOne",

"numCompsUsing": 1,

"parms": [

{

"parmType": "i",

"roleName": "input1"

},

{

"parmType": "i",

"roleName": "input2"

},

{

"parmType": "o",

"roleName": "output"

}

],

"props": {

"input1\_MISSING": "IGNORE",

"chooseHigher": "true",

"upperLimit": "999999999999.9",

"lowerLimit": "-999999999999.9",

"input2\_MISSING": "IGNORE"

}

}

### POST algorithm

The POST algorithm method requires a valid token. It takes a single OpenDCS Algorithm Record in JSON format, as described above for GET.

For creating a new record, leave algorithmId out of the passed data structure.

For overwriting an existing one, include the algorithmId that was previously returned. The algorithm in the database is replaced with the one sent.

### DELETE algorithm

The DELETE algorithm method requires a valid token.

Required argument algorithmid must be passed in the URL.

This operation will fail if the algorithm is currently being used by any computation records.

## Computation Methods

### GET computationrefs

Example: [http://localhost:8080/OHydroJson/computationrefs](http://localhost:8080/OHydroJson/algorithmrefs)

The returned structure is:

[

{

"algorithmId": 24,

"algorithmName": "AverageAlgorithm",

"computationId": 1,

"description": "",

"enabled": false,

"name": "Daily Ave [ ... )",

"processId": 5,

"processName": "compproc\_regtest"

},

{

"algorithmId": 24,

"algorithmName": "AverageAlgorithm",

"computationId": 2,

"description": "",

"enabled": false,

"name": "Daily Ave ( ... ]",

"processId": 5,

"processName": "compproc\_regtest"

},

{

"algorithmId": 24,

"algorithmName": "AverageAlgorithm",

"computationId": 3,

"description": "",

"enabled": false,

"name": "Daily Ave [ ... ]",

"processId": 5,

"processName": "compproc\_regtest"

},

…

]

### GET computation

Example:

[http://localhost:8080/OHydroJson/computation?computationid=4](http://localhost:8080/OHydroJson/algorithm?algorithmid=4)

Return data format:

{

"algorithmId": 24,

"algorithmName": "AverageAlgorithm",

"appId": 5,

"comment": "",

"computationId": 4,

"effectiveEndType": "No Limit",

"effectiveStartType": "No Limit",

"enabled": false,

"groupId": -1,

"lastModified": "2022-05-17T17:17:04.693Z[UTC]",

"name": "Daily Ave ( ... )",

"parmList": [

{

"algoParmType": "i",

"algoRoleName": "input",

"dataType": "Stage",

"dataTypeId": 48,

"deltaT": 0,

"duration": "0",

"interval": "15Minutes",

"paramType": "Inst",

"siteId": 1,

"siteName": "OKVI4",

"tsKey": 1,

"version": "raw"

},

{

"algoParmType": "o",

"algoRoleName": "average",

"dataType": "Stage",

"dataTypeId": 48,

"deltaT": 0,

"duration": "1Day",

"interval": "1Day",

"paramType": "Ave",

"siteId": 1,

"siteName": "OKVI4",

"tsKey": 5,

"version": "OO"

}

],

"props": {

"minSamplesNeeded": "1",

"aggUpperBoundClosed": "false",

"aggregateTimeZone": "UTC",

"average\_tsname": "HG-Ave-Open-Open",

"aggLowerBoundClosed": "false"

}

}

### POST computation

The POST computation method requires a valid token. It takes a single OpenDCS Computation Record in JSON format, as described above for GET.

For creating a new record, leave computationId out of the passed data structure.

For overwriting an existing one, include the computationId that was previously returned. The computation in the database is replaced with the one sent.

### DELETE computation

The DELETE computation method requires a valid token.

Required argument computationid must be passed in the URL.

## TSDB Properties Methods

### GET tsdb\_properties

Example: <http://localhost:8080/OHydroJson/tsdb_properties?token=1241234>

* The token argument is optional. If supplied it will reset the timer on the token.

The tsdb\_properties table in the database will be returned in a structure as follows:

{

"offsetErrorAction": "ROUND",

"storagePresentationGroup": "hydrodcs",

"api.datasource": "Cove-LRGS",

"allowDstOffsetVariation": "true"

}

### POST tsdb\_properties

The POST tsdb\_properties method requires a valid token. It takes one ore more properties in a structure as defined above for the GET method.

* Any property with the same name as one supplied will be overwritten by the passed value.
* If there is no property in the database with a matching name, a new property is added.
* To delete a property from the database, pass an empty string as the value.

## Engineering Unit Methods

### GET unitlist

Example: <http://localhost:8080/OHydroJson/unitlist>

* The token argument is optional. If supplied it will reset the timer on the token.

An array of data structures representing all known Engineering Units will be returnd as shown below.

[

{

"abbr": "$",

"family": "univ",

"measures": "Currency",

"name": "Dollars"

},

{

"abbr": "%",

"family": "univ",

"measures": "ratio",

"name": "percent"

},

{

"abbr": "1000 m2",

"family": "Metric",

"measures": "Area",

"name": "Thousands of square meters"

},

{

"abbr": "1000 m3",

"family": "Metric",

"measures": "Volume",

"name": "Thousands of cubic meters"

},

{

"abbr": "C",

"family": "Metric",

"measures": "Temperature",

"name": "Centigrade"

},

…

]

## Data Type Methods

### GET datatypelist

Examples:

<http://localhost:8080/OHydroJson/datatypelist>

<http://localhost:8080/OHydroJson/datatypelist?standard=cwms>

The returned data structure is an array of JSON data type objects:

[

{

"code": "72114",

"displayName": "CWMS:72114",

"id": 367,

"standard": "CWMS"

},

{

"code": "Address",

"displayName": "CWMS:Address",

"id": 368,

"standard": "CWMS"

},

{

"code": "Code-Channel",

"displayName": "CWMS:Code-Channel",

"id": 382,

"standard": "CWMS"

},

{

"code": "Code-DCPAddress",

"displayName": "CWMS:Code-DCPAddress",

"id": 372,

"standard": "CWMS"

},

{

"code": "Depth-Snow",

"id": 72,

"standard": "CWMS"

},

…

]

If the optional ‘standard’ argument is supplied, then only data types with the matching standard are returned. Otherwise all data types in the database are returned sorted by (standard, code).

## Retrieving Property Specs

GET propspecs?class=***FullyQualifiedClassName***

Many of the Java classes within OpenDCS maintain a set of properties that can alter the object’s behavior. This method allows the caller to get a list of acceptable properties for a given class.

For example, here are the properties accepted by the class decodes.db.Platform

GET propspecs?class=decodes.db.Platform

[

{

"description": "(default=0) Set to 1, 2, 3 for increasing levels of debug information when this platform is decoded.",

"dynamic": false,

"name": "debugLevel",

"type": "i"

},

{

"description": "Set to have this platform ignored during a specified season.",

"dynamic": false,

"name": "ignoreSeason",

"type": "e:Season"

},

{

"description": "Set to have this platform only processed during a specified season.",

"dynamic": false,

"name": "processSeason",

"type": "e:Season"

},

{

"description": "(default=3) For polled stations, this determines the order in which they will be polled (1 = highest priority = polled first)",

"dynamic": false,

"name": "pollPriority",

"type": "i"

}

]

An array of PropertySpec objects is returned in JSON format:

* “name” is the property name
* “description” is a description displayed to the user as a tooltip or in a dialog to set the property.
* “type” gives the code information about acceptable values for this property.
* “dynamic” is not used and should be ignored.

The “type” variable can be used by the code to offer a pull-down list or to limit the range of acceptable values. “type” will be one of the following:

|  |  |
| --- | --- |
| i | Integer – Value can only be a whole number. |
| n | Number – may be integer or floating point |
| b | Boolean – Value can be “true” or “false” |
| f | Filename |
| d | Directory Name |
| s | Free-form string |
| t | Time Zone Abbreviation – See the Java TimeZone doc for a list of acceptable abbreviations. |
| e:***RefListName*** | A pulldown list of values from a DECODES Reference List should be offered to the user (with a blank space at the top meaning ‘no selection’). See section 4.3 above for info on how to retrieve reference lists. |
| h | A host name or IP address |
| l | (That’s lower case L) indicates a long string that should be displayed in a multi-line text area with an optional scroll bar. But be aware that most of the property tables in the database limit a property value to 240 chars. |

The following class names currently support property specs. More may be added in the future:

* decodes.comp.CompResolver
* decodes.consumer.DataConsumer
* decodes.consumer.OutputFormatter
* decodes.datasource.DataSourceExec
* decodes.db.Platform
* decodes.db.PlatformSensor
* decodes.db.ConfigSensor
* decodes.dbeditor.RoutingSpecEditPanel
* decodes.routing.RoutingScheduler
* decodes.tsdb.CompAppInfo
* decodes.tsdb.algo.AW\_AlgorithmBase
* decodes.tsdb.compedit.AlgorithmEditPanel
* decodes.tsdb.ProcessEditPanel
* decodes.util.DecodesSettings
* lrgs.lrgsmain.LrgsConfig
* opendcs.opentsdb.OpenTsdbSettings

## Retrieve a Raw DCP Message

Example URLs for GET:

http://localhost:8080/OHydroJson/message?tmid=435107EA&tmtype=self-timed

http://localhost:8080/OHydroJson/message?tmid=300434063963670

The tmid argument is required. It must match a transport medium ID in a DCP Message. For GOES messages this is the 8-hex-digit GOES DCP Address. For Iridium messages this is the 15 digit IMEI. For Polled-DCP messages this is the name assigned to the logger.

The tmtype argument is optional. It’s primary use is to distinguish between GOES self-timed and goes random messages. It should either be “self-timed” or “random”.

The return data will be a JSON representation of the base64 encoded message. The most recent message will be returned:

{

"base64": "NDM1MTA3RUEyMjE4MTE1MjkxMkc0MC0wTk4wMzBXTjIwMDE1NGJCMUREW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hEW2hESG9ESGlESGRESFxESFZESFRESEtESERER3tER3ZER3JER2dER15ER1pER1RER1BER0pER0NERn1ERndERnFERmhERmBERllJQUJAQCA="

}

## Test Decoding

Example URL for HTTP POST method:

<http://localhost:8080/OHydroJson/decode?token=73168ed6c9c49870&script=ST>

As shown, the arguments must include:

* A valid security token
* The script name to use in decoding the message (if omitted, the first script in the config will be used).

The data sent in the POST request must be:

{

"config": ***{ config as returned by GET config described above}***

"rawmsg": ***{ Raw Message as returned by GET message described above }***

}

Here is an example request body:

{

"config": {

"configId": 2,

"configSensors": [

{

"absoluteMax": 30,

"absoluteMin": 0.009999999776482582,

"dataTypes": {

"SHEF-PE": "HG",

"CWMS": "Stage"

},

"properties": {},

"recordingInterval": 900,

"recordingMode": "F",

"sensorName": "Stage",

"sensorNumber": 1,

"timeOfFirstSample": 0

},

{

"absoluteMax": 251,

"absoluteMin": -0.10000000149011612,

"dataTypes": {

"SHEF-PE": "PC",

"CWMS": "Precip"

},

"properties": {},

"recordingInterval": 900,

"recordingMode": "F",

"sensorName": "Precip",

"sensorNumber": 2,

"timeOfFirstSample": 0

},

{

"dataTypes": {

"SHEF-PE": "VB",

"CWMS": "Volt-Battery"

},

"properties": {},

"recordingInterval": 3600,

"recordingMode": "F",

"sensorName": "Battery",

"sensorNumber": 3,

"timeOfFirstSample": 0

}

],

"description": "",

"name": "ROWI4",

"numPlatforms": 0,

"scripts": [

{

"dataOrder": "U",

"formatStatements": [

{

"format": "4x,8(f(s,b,3,1),f(s,b,3,2)),24x,f(s,b,1,3)",

"label": "st",

"sequenceNum": 0

}

],

"name": "ST",

"scriptSensors": [

{

"sensorNumber": 1,

"unitConverter": {

"a": 0.01,

"algorithm": "usgs-standard",

"b": 0,

"c": 1,

"d": 0,

"e": 0,

"f": 0,

"fromAbbr": "raw",

"toAbbr": "ft"

}

},

{

"sensorNumber": 2,

"unitConverter": {

"a": 0.01,

"algorithm": "usgs-standard",

"b": 0,

"c": 1,

"d": 0,

"e": 0,

"f": 0,

"fromAbbr": "raw",

"toAbbr": "in"

}

},

{

"sensorNumber": 3,

"unitConverter": {

"a": 0.3125,

"algorithm": "usgs-standard",

"b": 0,

"c": 1,

"d": 0.311,

"e": 0,

"f": 0,

"fromAbbr": "raw",

"toAbbr": "v"

}

}

]

}

]

},

"rawmsg": {

"base64": "Q0UyREQ2MzIyMjE5MjE0MDYyM0c0Ni0xTk4wNDlFTjIwMDA3OCJCU1RARnZAVX1ARm5ARnhAVX1ARm5ARnhAVX1ARm5ARnhAVX1ARm5ARndAVX1ARm5ARnZAVX1ARm5ARnhAVX1ARm5ARnhAVX1ARm9nIA=="

}

}

The raw data will be decoded according to the instructions in the passed config. The returned data will include log messages generated to trace the script execution, and the decoded data from the message. Here is the return data for the above request. Note that for each decoded value, the position within the raw message is given.

{

"logMessages": [

{

"priority": "INFO ",

"text": "Header type 'GOES' length=37",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:ModulationIndex=N",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:Length=78",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:Quality=N",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:Channel=49",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:FrequencyOffset=-1",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:Spacecraft=E",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:FailureCode=G",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:Time=07/11/2022 14:06:23 UTC",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:GPS=1",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:UplinkCarrier=N2",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:DcpAddress=CE2DD632",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "INFO ",

"text": " PM:SignalStrength=46",

"timeStamp": "2022-07-14T17:14:27.332Z[UTC]"

},

{

"priority": "DBG1 ",

"text": "After decoding there are 17 decoded samples.",

"timeStamp": "2022-07-14T17:14:27.349Z[UTC]"

}

],

"messageTime": "2022-07-11T14:06:23Z[UTC]",

"timeSeries": [

{

"sensorName": "Stage",

"sensorNum": 1,

"units": "ft",

"values": [

{

"rawDataPosition": {

"end": 7,

"start": 4

},

"time": "2022-07-11T14:00:00Z[UTC]",

"value": "4.38"

},

{

"rawDataPosition": {

"end": 13,

"start": 10

},

"time": "2022-07-11T13:45:00Z[UTC]",

"value": "4.3"

},

{

"rawDataPosition": {

"end": 19,

"start": 16

},

"time": "2022-07-11T13:30:00Z[UTC]",

"value": "14.05"

},

{

"rawDataPosition": {

"end": 25,

"start": 22

},

"time": "2022-07-11T13:15:00Z[UTC]",

"value": "4.4"

},

{

"rawDataPosition": {

"end": 31,

"start": 28

},

"time": "2022-07-11T13:00:00Z[UTC]",

"value": "4.3"

},

{

"rawDataPosition": {

"end": 37,

"start": 34

},

"time": "2022-07-11T12:45:00Z[UTC]",

"value": "14.05"

},

{

"rawDataPosition": {

"end": 43,

"start": 40

},

"time": "2022-07-11T12:30:00Z[UTC]",

"value": "4.39"

},

{

"rawDataPosition": {

"end": 49,

"start": 46

},

"time": "2022-07-11T12:15:00Z[UTC]",

"value": "4.3"

}

]

},

{

"sensorName": "Precip",

"sensorNum": 2,

"units": "in",

"values": [

{

"rawDataPosition": {

"end": 10,

"start": 7

},

"time": "2022-07-11T14:00:00Z[UTC]",

"value": "14.05"

},

{

"rawDataPosition": {

"end": 16,

"start": 13

},

"time": "2022-07-11T13:45:00Z[UTC]",

"value": "4.4"

},

{

"rawDataPosition": {

"end": 22,

"start": 19

},

"time": "2022-07-11T13:30:00Z[UTC]",

"value": "4.3"

},

{

"rawDataPosition": {

"end": 28,

"start": 25

},

"time": "2022-07-11T13:15:00Z[UTC]",

"value": "14.05"

},

{

"rawDataPosition": {

"end": 34,

"start": 31

},

"time": "2022-07-11T13:00:00Z[UTC]",

"value": "4.4"

},

{

"rawDataPosition": {

"end": 40,

"start": 37

},

"time": "2022-07-11T12:45:00Z[UTC]",

"value": "4.3"

},

{

"rawDataPosition": {

"end": 46,

"start": 43

},

"time": "2022-07-11T12:30:00Z[UTC]",

"value": "14.05"

},

{

"rawDataPosition": {

"end": 52,

"start": 49

},

"time": "2022-07-11T12:15:00Z[UTC]",

"value": "4.38"

}

]

},

{

"sensorName": "Battery",

"sensorNum": 3,

"units": "V",

"values": [

{

"rawDataPosition": {

"end": 77,

"start": 76

},

"time": "2022-07-11T14:00:00Z[UTC]",

"value": "12.4985"

}

]

}

]

}

# Time Series Methods

## GET Time Series Catalog

The tscatalog method can be used to return the entire, or subset of, the time series catalog that is known to the server.

Examples:

http://localhost:8080/OHydroJson/tscatalog?site=BFD,CHCR&param=Elev

http://localhost:8080/OHydroJson/tscatalog?param=Stage&interval=1Hour,4Hours

Without arguments, it will return the entire catalog of time series. The following option arguments may be included:

* site: comma-separated list of site names. Only TSIDs from specified site(s) will be returned.
* param: comma-separated list of parameter values
* interval: comma-separated list of interval values
* active: boolean value (e.g. yes, no, true, false, T, F, Y, N).
* tsids: comma-separated list of complete time series IDs. If present, this overrides the site, param, interval, and active filters.
* token: Not required, but if included, the token lastUsed time will be updated.

The arguments may be combined for an increasingly fine filter.

The return JSON value contains time series identifiers according to the original HydroJSON specification. The format is as follows:

{

"BFD": {

"HUC": "",

"active\_flag": "T",

"coordinates": {

"datum": "WGS84",

"latitude": 42.4278,

"longitude": -72.0261

},

"elevation": {

"accuracy": 0,

"datum": "NGVD29",

"method": "",

"value": 234.7

},

"location\_id": "BFD",

"location\_type": "",

"name": "BFD",

"responsibility": "",

"time\_format": "%Y-%m-%dT%H:%M:%S%z",

"timeseries": {

"BFD.Precip.Inst.15Minutes.0.DCP-raw": {

"active\_flag": "T",

"count": 0,

"duration": "0",

"interval": "15Minutes",

"notes": "BFD.Precip.Inst.15Minutes.0.DCP-raw",

"param": "Precip",

"sigfig": 3,

"units": "in"

},

"BFD.Elev-Pool.Inst.15Minutes.0.DCP-raw": {

"active\_flag": "T",

"count": 0,

"duration": "0",

"interval": "15Minutes",

"notes": "BFD.Elev-Pool.Inst.15Minutes.0.DCP-raw",

"param": "Elev-Pool",

"sigfig": 3,

"units": "ft"

}

},

"timezone": "America/New\_York",

"tz\_offset": 0

}

}

## GET Time Series Data

The tsdata method extends the tscatalog method. The same filtering arguments described for tscatalog may be used here. In addition, you may specify a time range for data retrieval by using the since and until arguments.

Example:

http://localhost:8080/OHydroJson/tsdata?site=BFD,CHCR&active=true&since=2011/021/22:00&until=2011/021/23:30

The since and until arguments may have any of the following formats:

now-1day The word “now” minus an increment times a unit. Examples: now-1day, now-5hours, now-1week, etc.

now The current time that the web service call was made.

YYYY/DDD/HH:MM:SS A complete Julian Year, Day-of-Year, and Time

YYYY/DDD/HH:MM Seconds omitted means zero.

DDD/HH:MM:SS Assume current year

DDD/HH:MM

HH:MM:SS Assume current day

HH:MM

If ‘since’ is omitted, then the earliest values in storage are included. If ‘until’ is omitted, then the latest values are included.

Data is returned in a format extended from tscatalog described above. After the “units” field will be a set of values. For each value, the date/time, value, and flags are provided.

For the query:

http://localhost:8080/OHydroJson/tsdata?tsid=BFD.Precip.Inst.15Minutes.0.DCP-raw,BFD.Elev-Pool.Inst.15Minutes.0.DCP-raw&since=2011/22/00:00&until=2011/22/01:00

Here is the response:

{

"BFD": {

"HUC": "",

"active\_flag": "T",

"coordinates": {

"datum": "WGS84",

"latitude": 42.4278,

"longitude": -72.0261

},

"elevation": {

"accuracy": 0,

"datum": "NGVD29",

"method": "",

"value": 234.7

},

"location\_id": "BFD",

"location\_type": "",

"name": "BFD",

"responsibility": "",

"time\_format": "%Y-%m-%dT%H:%M:%S%z",

"timeseries": {

"BFD.Precip.Inst.15Minutes.0.DCP-raw": {

"active\_flag": "T",

"count": 5,

"duration": "0",

"end\_timestamp": "2011-01-22T01:00:00",

"interval": "15Minutes",

"max\_value": 15.51,

"min\_value": 15.51,

"notes": "BFD.Precip.Inst.15Minutes.0.DCP-raw",

"param": "Precip",

"sigfig": 3,

"start\_timestamp": "2011-01-22T00:00:00",

"units": "in",

"values": [

[

"2011-01-22T00:00:00",

15.51,

0

],

[

"2011-01-22T00:15:00",

15.51,

0

],

[

"2011-01-22T00:30:00",

15.51,

0

],

[

"2011-01-22T00:45:00",

15.51,

0

],

[

"2011-01-22T01:00:00",

15.51,

0

]

]

},

"BFD.Elev-Pool.Inst.15Minutes.0.DCP-raw": {

"active\_flag": "T",

"count": 5,

"duration": "0",

"end\_timestamp": "2011-01-22T01:00:00",

"interval": "15Minutes",

"max\_value": 776.94,

"min\_value": 776.94,

"notes": "BFD.Elev-Pool.Inst.15Minutes.0.DCP-raw",

"param": "Elev-Pool",

"sigfig": 3,

"start\_timestamp": "2011-01-22T00:00:00",

"units": "ft",

"values": [

[

"2011-01-22T00:00:00",

776.94,

0

],

[

"2011-01-22T00:15:00",

776.94,

0

],

[

"2011-01-22T00:30:00",

776.94,

0

],

[

"2011-01-22T00:45:00",

776.94,

0

],

[

"2011-01-22T01:00:00",

776.94,

0

]

]

}

},

"timezone": "America/New\_York",

"tz\_offset": 0

}

}

## POST Convert Netlist from “.nl” File

This POST method is sent the contents of an LRGS “.nl” file as plain text and returns a JSON data structure representation of the file.

{"items":{

"12345678":{"description":"Some Big Long Description",

"platformName":"SomeName","transportId":

"12345678"},"87654321":{"description":"Description Two",

"platformName":"NameNumber2","transportId":"87654321"}

}

}