CSI Datalogger Web Service API

Jon Trauntvein

Version 1.00.11

22 April 2021

Table of Contents

1. Introduction

The Datalogger Web Services API is a collection of HTTP commands that can be used to query a datalogger for its data and meta-data and to control that datalogger. This API is implemented by the CSI Web Server product as well as the CR1000, CR800, CR3000, and CR6 dataloggers. This document is intended to document the various HTTP commands and formats that can be used in this API.

1.1. Commands Common Features

All of the transactions in the Datalogger Web Service API follow certain patterns. For the CSI Web Server product, the published project that responds to a request will depend upon the path specified in the request URL. For instance, if I have a project published to the `oft` sub-directory in the web server's home directory and a received request specified that path, the servlet that handles that request will be associated with that published path.

For dataloggers, the path specified in the request will not matter. The datalogger will respond the same way regardless of the path that was specified.

Every request that can be sent as a part of the Datalogger Web Services API must contain a URL parameter named `command` that will specify the name of the operation to be performed. Every request can optionally specify a `format` URL parameter that will specify the format in which the client expects a response.

1.1.1. `uri` Parameter

Many requests can specify a `uri` parameter that can specify the path of a data source, station, table, or variable that is effected by the request. How this parameter value is interpreted will depend upon the web server and the data source type. This parameter must be formatted according to the following syntax:

```
uri          := source-name [ ":" data-address ].
data-address := ln-address |
                db-address |
                file-address |
                http-address |
                logger-address.
ln-address   := station [ "." table [ "." field-name ]].
db-address   := table [ "." field-name ].
file-address := table [ "." field-name ].
http-address := table [ "." field-name ].
logger-address := table [ "." field-name ].
field-name   := field [ "(" subscript
                        { "," subscript } ")" ].
```

If any of the components of a data source URI contains a period character ('.'), that character must be escaped using a backslash ('\') in the URI. Note that this same syntax is used in RTMC expressions to identify data fields, stations, and tables.

1.1.2. `format` Parameter

Most commands will recognise a `format` parameter that optionally specifies the format of the response message. The following values are recognised by most commands:

`html` Specifies the response should be formatted as HTML. If the `format` parameter is not specified, the format of the response will automatically be selected as HTML.

`xml` Specifies that the response should be formatted as an XML document. The structure of this document will depend upon the `command` parameter.

`json` Specifies that the response should be formatted as a JSON document. The structure of this document will depend upon the `command` parameter. This option is generally the most useful when the response needs to be parsed by a machine.

1.2. Authentication

Because most HTTP transactions take place over a publicly available TCP interface and can expose control services particularly, it is necessary to secure access to the web server. The datalogger web services API uses Basic Access Authentication (see http://en.wikipedia.org/wiki/Basic_access_authentication) in order to control access. In addition to this, it is possible to use the API using HTTPS. In addition to this, it may be possible to use Digest Access Authentication (see http://en.wikipedia.org/Digest_access_authentication) with minimal changes to the services.

1.2.1. `.csipasswd` File

Authentication access information will be encoded in a `.csipasswd` that will be stored on the web server. The dataloggers will maintain this file on their `CPU:` storage device. The CSI Web Server application will maintain a root password file but will also allow individual projects published to the web server (in sub-directories) to be used. The `.csipasswd` file is very similar to the `.htaccess` files used by Apache web servers. The `.csipasswd` file is a text file that must conform to the following format:

```
csipasswd := realm "\r\n" { account "\r\n" }.
account   := account-name ":" encrypted-passwd
             [ ":" access-level ].
access-level := ("0" | "1" | "2" | "3").
```

`realm` Specifies the string that the browser will use to identify the site when it presents the logon dialogue.

`account-name` Specifies the user's account name.

`encrypted-passwd` Specifies the password associated with the account encrypted using the CSI signature based encryption.

`access-level` An encoded integer that identifies that user's access level. With the exception of value zero, these access levels should correspond with the access levels assigned by datalogger security. The following values are defined:

0 All access denied

1 All access allowed

2 Set variables allowed

3 Read-only access

`.csipasswd` files can be created on the datalogger using the `Edit .csipasswd File` button that can be found in the Device Configuration Utility's `Deployment/Network Services` panel for dataloggers. It can be edited for the CSI web server using the CSI Web Admin utility.

If the client sends a request that does not contain authentication parameters in its header, the server will default to whatever security level is assigned to the `anonymous` account. This account can be disabled by assigning an access level of zero or removing that account.

If a client sends an HTTP request that does not contain authentication parameters in its header or the authentication parameters are invalid (or reference a disabled account), the web server will respond with an HTTP response code of `401 Authorization Required`.

1.2.2. Using Bearer Authorisation and Access Tokens

Starting with version 1.07 of the CSI Web Server and also with newer dataloggers, a web client can also choose to use `Bearer` type authentication in the `Authorization` HTTP header followed by a an access token that can be obtained using the services described in Section 2.7, “Commands Associated with Access Tokens”. An access token is an opaque string that is generated by the web server using client-provided user credentials. This access token can be sent by the client in the HTTP header of a request similar to the following:

```
Authorization: Bearer eyJhbGciOiAiSFMyNTYiLCJ0eXAiOiAiSldUIn0.
                       eyJpYXQiOiAxNjAyNTY1OTkwLCJleHAiOiAxNjAy
                       NTY5NTkwLCJ1c2VyIjogImpvbiIsImFjY2VzcyI6
                       IDF9.8WdGw3tm1FSltLiXtf5gHMRjOHp39Q_DLHI
                       8Dsyr1ZY
```

Note that the line breaks were introduced for readability on the page. The access token sent in the actual request would be formatted all on the same line in the header.

The format of this bearer token is JWT (JSON Web Token) and is described in RFC 7519 [https://tools.ietf.org/html/rfc7519]. It consists of a header, body, and signature that are all Base64 URL

encoded and separated by periods. To the web client, these tokens can be treated as opaque strings that are provided as authorisation for one or more HTTP requests. To the web server, the token header and body contains information including user identification that has been signed by the web server in such a way that the token cannot be changed without invalidating the digest at the end of the token which the server will use to validate the token.

A client first obtains an access token by invoking the server's `GetAccessToken` command (see Section 2.7.1, "`GetAccessToken` Command"). If this request succeeds, the server will return a JSON structure that provides an `access_token` and `refresh_token` properties aas well as properties that report the expiration interval for both of these properties. The client can persist these tokens by storing a copy of this structure in local storage. The first request to obtain these tokens requires that the client provide user credentials (user name and password) in order to generate the first set of tokens. Once this transaction succeeds, the client can refresh access tokens by periodically posting the `refresh_token` in the `RefreshAccessToken` command (see Section 2.7.2, "`RefreshAccessToken` Command"). The response to this request will also be a JSON structure that will contain newly issued access and refresh tokens.

2. Datalogger Web Services API Commands

The commands and responses in the Datalogger Web Services API can be divided up into the following categories:

2.1. Data Collection Related Commands

A client can collect data from the web server by specifying `DataQuery` as the value for the `command` parameter in the URI. The client must provide other parameters that will specify the data to be returned, starting positions, order options, and format.

2.1.1. `DataQuery` Command

In order to successfully send a `DataQuery` command, the client must authenticate with an account that has a minimum access level of `read-only`.

The `DataQuery` command supports the following URI parameters:

`command` Must be set to a value of `DataQuery` in order to specify that the request is a data query command.

`uri` Specifies the address of the data to be returned. This parameter must reference either a table or a data field value. It must conform to the syntax described in Section 1.1.1, "`uri` Parameter".

`mode` Specifies the nature of the data query. The following values are supported:

- `most-recent` Specifies that up to the number specified by `p1` of most recent records logged in the table should be returned.
- `since-time` Specifies all of the records logged since the time stamp specified in `p1` should be returned.
- `since-record` Specifies that all records with a record number greater than or equal to the value specified in `p1` should be returned. If that record number is not present in the table, data should returned starting with the oldest record in the table. Starting with CSI Web Server version 1.0.50, the `p2` parameter can optionally specify a start time stamp as well. This feature will provide the server more information to choose the starting point in the case of data file and database data sources where the record number can be repeated.
- `date-range` Specifies that all data contained in the half-open time interval specified by the `p1` and `p2` parameters which are interpreted as time stamps. Records will be returned that have a time stamp greater than or equal to the time stamp specified by `p1` and less than the time stamp specified by `p2`.
- `backfill` Specifies that all records starting with the first record that has a time stamp greater than or equal to the time stamp of the newest record less the time interval specified by `p1` in seconds.

`p1` Specifies qualifying information based upon the value of the `mode` parameter. The interpretation of this parameter varies as follows:

- `most-recent` — Specifies the maximum number of records to return.
- `since-record` — Specifies the record number at which the server should start sending records.
- `since-time`, `date-range` — Specifies the starting record time stamp. The first record that has a time stamp that is greater than or equal to this value will be selected as the starting point.
- `backfill` — Specifies a time interval in units of seconds that should be subtracted from the time stamp of the newest table record in order to find the starting record.

`p2` Specifies further qualifying information for some values of the `mode` parameter. The interpretation of this parameter varies as follows:

- `date-range` — Specifies the ending time stamp of the half-open time interval. The records returned will have a time stamp less than this value.
- `since-record` — Optionally specifies a starting time stamp to be used with data sources, such as data files and databases, that have the possibility of having duplicate record numbers.
- All others — This parameter will be ignored for any other value of `mode`.

`format` Optionally specifies the expected format of the server response. If this parameter is not specified, a value of `html` will be assumed by the server. The following values are supported:

- `html` — Specifies that the data will be formatted as an HTML table and that the `Content Type` field of the HTTP response must be set to `text/html`.
- `json` — Specifies that the data will be returned in a JSON structure that conforms to the CsiJSON format (see chapter 9 of http://engsoft.campbellsci.com/protocols/csi_file_formats.pdf). The server must specify the `Content Type` field in the HTTP response as `application/json`.

  When the `json` format is specified, there are certain features which become available:

  - The server may choose to break up the response into multiple queries. It will indicate whether there are more records available using the `more` member in the CsiJSON response structure.

- The server will send a table definition signature in the `head.signature` of the CsiJSON response. If the client sends this signature in `headsig` parameter of the next request and this signature matches the server's, the server will not include the `head.environment` or `head.fields` members of the response.

`toa5` Specifies that the data should be returned as comma-separated values with an extended header. The return format must conform to that specified in chapter 7 of http://engosoft.campbellsci.com/protocols/csi_file_formats.pdf. The server must specify the `Content Type` field of the HTTP response as `text/csv`.

`tob1` Specifies that the data should be returned as binary with a comma-separated header. The format of the response content must conform to that specified in chapter 12 of http://engsoft.campbellsci.com/protocols/csi_file_formats.pdf. The `Content Type` field of the HTTP response must be set to `binary/octet-stream`.

`xml` Specifies that the data should be formatted as an XML structure as described in chapter 8 of http://engsoft.campbellsci.com/protocols/csi_file_formats.pdf. The `Content Type` field of the HTTP response must be set to a value of `text/xml`.

`nextpoll` Optionally specifies the time interval, in seconds, in the future at which the web client expects to repeat this query. This parameter will be ignored by the datalogger but it will be used by the CSI web server to cache the associated data source request if it can be cached. If this parameter is not specified or it is specified with a value of 0xFFFFFFFF (the default), the data source will not be cached.

`transaction` Optionally specifies the transaction number that the server will use to set the `transaction` identifier in the CSIJson format when the `format` parameter is set to `json`. This is useful inside of a web client in keeping track of responses to various pending requests. This parameter will be ignored for any other value of the `format` parameter.

`headsig` Optionally specifies a data header signature returned from a previous request formatted as `json`. If the value of this parameter matches the web server's value, the web server will not include the `environment` or `fields` fields in the response data structure. This parameter will be ignored for any other value of the `format` parameter. The server will return the expected value in the `head.signature` member of any CSIJson response.

`refresh` Optionally specifies the minimum interval, in seconds, at which the table specified by `uri` should be polled for LoggerNet data sources. This parameter will be ignored by the datalogger and will also be ignored by the CSI Web Server for tables that are not from LoggerNet sources. If this parameter is specified and the table identified is associated with a LoggerNet source, the

web server will perform selective manual polls for the table at the specified interval.

`order` Specifies the order in which the records should be reported in the response from the CSI Web Server. This parameter is ignored by the datalogger. The following values are defined:

- `real-time` Specifies that the server will skip over historic records in favour of most recently logged records.
- `collected` Specifies that the records should be returned in the order in which they were collected by LoggerNet. This order can differ from logged order when one-way or data advise data collection is used in conjunction with hole collection.
- `logged-with-holes` Specifies that the data should be reported in the order in which it was logged by the datalogger. If there are holes that have not yet been collected by LoggerNet, newer data will not be returned until those holes have been collected or they are deemed no longer collectible by LoggerNet.
- `logged-without-holes` Specifies that the data should be reported in the order in which it was logged by the datalogger. If there are uncollected holes, those records will be skipped in favour of more recent records.

2.1.2. `DataQuery` Response

The format of the response as well the `Content Type` field of the HTTP response will depend upon the value of the `format` parameter in the HTTP request. Since one of the reasons for using the `DataQuery` command is to be able to store the data into a file, the server must specify the `Content-Disposition` field in the response header. An example of this would be as follows:

```
Content-Disposition:filename=OneDay.json
```

2.1.3. `DataQuery` Command Examples

We will present examples of commands and responses in various formats for the same data. Note that the URL examples have been broken up to fit on the printed page. If used, the URL must appear all on one line.

2.1.3.1. HTML Format Example

The following URL will produce an HTML format output:

```
http://jtrauntvein-mpw/engsoft?command=DataQuery&
uri=localhost:cr1000.one_day.temp_degf_avg&
mode=most-recent&
p1=1&
format=html
```

This command will produce the following output from the CSI Web Server:

```
<!DOCTYPE HTML>
<html> <head>
<title>Table Display</title>
</head>
<body>
<h1>Table Name: one_day</h1>
<table border='1'>
<tr>
<th nowrap='1'>Time Stamp</th>
<th nowrap='1'>Record</th>
<th nowrap='1'>temp_degf_Avg</th>
</tr>
<tr>
<td nowrap='1'>2014-11-25 00:00:00</td>
<td>69</td>
<td nowrap='1'>72.40345</td>
</tr>
</table>
</body>
</html>
```

2.1.3.2. JSON Format Example

The following URL can produce an output in JSON:

```
http://jtrauntvein-mpw/engsoft?command=DataQuery&
uri=localhost:cr1000.one_day.temp_degf_avg&
```

```
mode=most-recent&
p1=1&
format=json
```

This will produce the following output:

```
{
  "head": {
    "transaction": 0,
    "signature": 64455,
    "environment": {
      "station_name": "cr1000",
      "table_name": "one_day"
    },
    "fields": [
      {
        "name": "temp_degf_Avg",
        "type": "xsd:float",
        "units": "DegF",
        "process": "Avg",
        "settable": false
      }
    ]
  },
  "data": [
      {
        "no": 69,
        "time": "2014-11-25T00:00:00",
        "vals": [ 72.40345 ]
      }
    ],
    "more":  false
}
```

We can condense the output from the server by specifying the header signature in the URL:

```
http://jtrauntvein-mpw/engsoft?command=DataQuery&
uri=localhost:cr1000.one_day.temp_degf_avg&
mode=most-recent&
p1=1&
format=json&
```

```
headsig=64455
```

This URL will generate the following response:

```
{
  "head": {
    "transaction": 0,
    "signature": 64455
  },
  "data": [
      {
        "no": 69,
        "time": "2014-11-25T00:00:00",
        "vals": [ 72.40345 ]
      }
    ],
    "more":  false
}
```

2.2. Browse Server Symbols

Symbols represent meta-data or information about the data that is available on the server. The `BrowseSymbols` command can be used to get lists of symbols: data sources, stations, tables, and fields available on the server.

2.2.1. `BrowseSymbols` Command

The `BrowseSymbols` command can be used to obtain a list of child symbols to the symbol that is identified in the optional `uri` parameter. This command requires a minimum access level of `read-only` to succeed. If a client authorises with an account that does not have at least `read-only` permission, the server must respond with a `401 Authorization Required` in its HTTP response. The `BrowseSymbols` command supports the following parameters:

`command` `BrowseSymbols` must be specified in order to browse symbols.

`uri` Optionally specifies the identifier for the parent symbol for which the response should list children. This value must conform to the syntax specified in Section 1.1.1, "`uri` Parameter".

`format` Optionally specifies the format of the response. If not specified, the format will default to `html`. Supported values include `html`, `xml`, and `json`.

2.2.2. `BrowseSymbols` Response

The format of the response will depend upon the value of the `format` parameter of the `BrowseSymbols` command. Regardless of the format, however, the following information will be returned for each symbol:

`uri` Specifies the unique identifier for the child symbol. This value must conform to the syntax described in Section 1.1.1, "`uri` Parameter". If the client specifies a `uri` value for a symbol that does not exist, the server must return an empty list of child symbols.

`name` Specifies the name of the symbol. This could be a data source name, a station name, a table name, or a field name.

`type` Specifies a numeric code for the type of symbol being represented. The following values are defined:

1 LoggerNet Data Source

2 Data File Data Source

3 Database Data Source

9 HTTP Data Source

4 LoggerNet Station

5 LoggerNet Statistics Broker

6 Table

7 Array

8 Scalar

`is_enabled` Specifies a boolean value that is true if the symbol is enabled for scheduled collection. This applies mostly to LoggerNet data sources.

`is_read_only` Specifies a boolean value that is set to true if the symbol is considered to be read-only. A value of false will indicate that the symbol value can be changed using the `SetValueEx` command.

`can_expand` Specifies a boolean value if this symbol represents something that can be expanded with another `BrowseSymbols` request.

2.2.2.1. HTML Response Format

If the `format` parameter of the `BrowseSymbols` command is set to `html` or it is not specified, the symbols will be output in an HTML table structure with one row per symbol. The following is an example of an output:

```
<!DOCTYPE HTML>
<html> <head>
<title>BrowseSymbols Response</title>
</head>

<body>
<h1>BrowseSymbols Response</h1>

<table border="1">
  <tr>
    <th>name</th>
    <th>uri</th>
    <th>type</th>
    <th>is_enabled</th>
    <th>is_read_only</th>
    <th>can_expand</th>
  </tr>
  <tr>
    <td>CR1000</td>
    <td>localhost:CR1000</td>
    <td>4</td>
    <td>true</td>
    <td>true</td>
    <td>true</td>
  </tr>
```

```
  <tr>
    <td>__Statistics__</td>
    <td>localhost:__Statistics__</td>
    <td>5</td>
    <td>true</td>
    <td>true</td>
    <td>true</td>
  </tr>
</table>

</body></html>
```

2.2.2.2. XML Response Format

If the `format` parameter to the `BrowseSymbols` command is set to a value of `xml`, the output will be formatted as an XML data structure with a `BrowseSymbolsResponse` root element and each child of that root representing a child symbol. The symbol attributes will be represented with XML attributes. The following is an example XML output:

```
<?xml version="1.0" encoding="UTF-8"?>
<BrowseSymbolsResponse>
  <symbol can_expand="true"
          is_enabled="true"
          is_read_only="true"
          name="CR1000"
          type="4"
          uri="localhost:CR1000" />
  <symbol can_expand="true"
          is_enabled="true"
          is_read_only="true"
          name="__Statistics__"
          type="5"
          uri="localhost:__Statistics__" />
</BrowseSymbolsResponse>
```

2.2.2.3. JSON Response Format

If the `format` parameter of the `BrowseSymbols` command is set to a value of `json`, the output will be structured as an object that contains an array of child objects named `symbols`. Each child object in that array will represent a child symbol. The following is an example JSON output:

```
{
  "symbols": [
    {
      "name": "CR1000",
      "uri": "localhost:CR1000",
      "type": 4,
      "is_enabled": true,
      "is_read_only": true,
      "can_expand": true
    },
    {
      "name": "__Statistics__",
      "uri": "localhost:__Statistics__",
      "type": 5,
      "is_enabled": true,
      "is_read_only": true,
      "can_expand": true
    }
  ]
}
```

2.3. Control (Set Variable)

The `SetValueEx` command can be used to set the value of a variable on the server. Dataloggers also support an older `SetValue` command but that version does not recognise HTTP authorisation or other URL parameters. In order to use the `SetValueEx` command, the client must authenticate using an account with at least `read/write` privileges assigned. If the authorisation account does not have this privilege, the server must respond with `401 Authorization Required`.

2.3.1. `SetValueEx` Command

The `SetValueEx` command is used to set the value of a variable identified by `uri` parameter and to return the result. Since this command has the effect of changing the data being reported, it should be performed using the `POST` HTTP method. This command recognises the following parameters:

`command` The `command` parameter must be set to a value of `SetValueEx`.

`uri` Identifies the value that should be set. This URI must identify a changeable variable in order for this transaction to succeed. This value must conform to the syntax described in Section 1.1.1, "`uri` Parameter".

`value` Specifies a string that represents the value to be set.

`format` Specifies the expected format of the response. This can be one of `html`, `xml`, or `json`. If this parameter is not specified, it will default to `html`.

2.3.2. `SetValueEx` Response

Regardless of the value of the `format` parameter, there are certain fields that will be returned. These fields are as follows:

`outcome` Specifies a numeric code that identifies the outcome of the transaction. The following values are defined:

0. An unrecognised failure occurred.
1. The variable was set (success)
2. The data source connection failed.
3. LoggerNet logon failed (LoggerNet sources only)
4. Blocked by LoggerNet security (LoggerNet sources only)
5. The column is read-only
6. Invalid table name specified
7. Invalid column name specified
8. Invalid column data type

9. Invalid column subscript
10. Datalogger communication failed
11. Datalogger communication is disabled (LoggerNet sources only)
12. Blocked by datalogger security
13. Invalid LoggerNet table definitions
14. Invalid LoggerNet device name
15. Invalid web client authorisation

`description` Specifies a text description of the outcome code.

2.3.2.1. HTML Response Format

The HTML response will be formatted as a table where each field is represented on a separate row. An example of this format follows:

```
<!DOCTYPE HTML>
<html> <head>
  <title>SetValueExResponse</title>
</head>

<body>
<h1>SetValueExResponse</h1>

<table>
  <tr>
    <td>outcome</td>
    <td>1</td>
  </tr>
  <tr>
    <td>description</td>
    <td>The variable was set</td>
  </tr>
</table>

</body> </html>
```

2.3.2.2. XML Response Format

The XML response format will consist of a single root element with the response parameters formatted as XML attributes. The following specifies an example of this format:

```
<?xml version="1.0" encoding="UTF-8"?>
<SetValueExResponse description="The variable was set"
                    outcome="1" />
```

2.3.2.3. JSON Response Format

The JSON response format will consist of an object with a `outcome` and `description` member. The following is an example of this format:

```
{
  "outcome": 1,
  "description": "The variable was set"
}
```

2.3.3. `CheckAuthorization` Command

The `CheckAuthorization` command can be used by the client to get the access level that the server has assigned for the provided authorisation parameters. Unlike other web API commands, this command requires that the authorisation be specified in the HTTP header. If there is no authorisation, the server must respond with `401 Authorization Required`. The command recognises the following parameters:

`command` The `command` parameter must be set to a value of `CheckAuthorization`.

`uri` Optionally identifies the station for which the authorisation value should be checked. This value must conform to the syntax described in Section 1.1.1, "`uri` Parameter". If this parameter is omitted, the server will evaluate the client's local access.

`format` Specifies the expected format of the response. This can be one of `html`, `xml`, `json`, or `html`. If this parameter is not specified, the default will be `html`.

`anonymous` Specifies a boolean value (`true` or `false`) that, if set, will return the access level associated with the `anonymous` account.

2.3.4. `CheckAuthorization` Response

Regardless of the value of the `format` parameter, the response will always contain an access level entitled `authorization`. This value will have the following values:

0. No access allowed

1. All access allowed
2. Read/Write access allowed
3. Read-Only access allowed

99.An error occurred in determining access level

2.3.4.1. HTML Response Format

The response to the `CheckAuthorization` command will be formatted as HTML when the `format` parameter is set to `html` or when the `format` parameter is not set. An example of this format follows:

```
<html>
  <head>
    <title>CheckAuthorizationResponse</title>
  </head>
  <body>
    <h1>CheckAuthorizationResponse</h1>
    <table>
      <tr>
        <td>authorization</td>
        <td>1</td>
      </tr>
    </table>
  </body>
</html>
```

2.3.4.2. JSON Response Format

The response to the `CheckAuthorization` command will be formatted as a JSON document when the `format` parameter is set to `json`. An example of this format follows:

```
{
  "authorization": 1
}
```

2.3.4.3. XML Response Format

The response to the `CheckAuthorization` command will be formatted as an XML document when the `format` parameter is set to `xml`. An example of this format follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<CheckAuthorizationResponse authorization="1"/>
```

2.4. Station/Server Time Related Commands

2.4.1. `ClockCheck` Command

The `ClockCheck` command can be used to read the clock from the web server, a LoggerNet server through a LoggerNet data source, or a datalogger. This command requires a minimum access level of `read-only` and should be accessed using the HTTP `GET` method. If the user's account does not have `read-only` privileges, the server must respond with a `401 Authorization Required` HTTP response.

The `ClockCheck` command recognises the following URI parameters:

`command` This parameter must be set to `ClockCheck`.

`uri` Optionally specifies the data source URI for the data source or station to check. If this parameter is not specified, the web server system time will be returned. This value must conform to the syntax specified in Section 1.1.1, "`uri` Parameter".

`format` Optionally specifies the expected format of the response. This can be one of `html` (the default if not specified), `xml`, or `json`.

2.4.2. `ClockCheck` Response

The response to the `ClockCheck` command will be as an HTML document, and XML document, or a JSON document depending upon the value of the `format` parameter of the command URI. Regardless of the format, the response will have the following parameters:

`outcome` Specifies a numeric code that describes the outcome of the command. The following values are defined:

1. The clock was checked.
2. The clock was set (LoggerNet may combine a new clock check transaction with an existing clock set transaction).
3. The LoggerNet session failed.
4. Invalid LoggerNet logon.
5. Blocked by LoggerNet security.
6. Communication with the specified station failed.
7. Communication with the specified station is disabled.
8. Blocked by datalogger security.
9. Invalid LoggerNet station name.
10. The LoggerNet device is busy.

11.The value specified by `uri` does not reference an object that has a clock.

`time` Specifies the current value of the server or datalogger real-time clock. This parameter will only be present if the value of `outcome` is set to one or two.

`description` Specifies a text description of the `outcome` parameter.

2.4.2.1. HTML Response Format

When the `format` option is set to `html` or is not specified, the response will be in the form of an HTML table where each response field is on its own row. The following example demonstrates this format:

```
<html>

<head>
<title></title>
</head>
<body>
<table border="1">
<tr>
<td>outcome</td><td>1</td>
</tr>
<tr>
<td>time</td><td>2014-11-26T12:39:23.252</td>
</tr>
<tr>
<td>description</td><td>The clock was checked</td>
</tr>
</table>
</body>
</html>
```

2.4.2.2. XML Response Format

When the `format` parameter is set to XML, the response will be formatted as an XML document with a single root element named `ClockCheckResponse` and with each parameter specified as an attribute. The following example demonstrates this format:

```
<?xml version="1.0" encoding="UTF-8"?>
<ClockCheckResponse description="The clock was checked"
                    outcome="1"
```

```
                        time="2014-11-26T12:44:25.629" />
```

2.4.2.3. JSON response Format

When the `format` parameter is set to `json`, the response will be formatted as a JSON document with a single object that has members named for each response parameter. The following example demonstrates this format:

```
{
  "outcome": 1,
  "time": "2014-11-26T12:49:11.879",
  "description": "The clock was checked"
}
```

2.4.3. `ClockSet` Command

The `ClockSet` command can be used to adjust the real time clock of a datalogger. This command requires a minimum of `read/write` access. If the command is attempted using a lower authorisation, the server must respond with a `401 Authorization Required` HTTP response. The command recognises the following URL parameters:

`command` The value of `command` must be `ClockSet` in order to set the clock.

`uri` Specifies the data source URI that identifies the datalogger that should have its clock set. This parameter is required for LoggerNet or HTTP data sources. The datalogger will ignore this parameter when acting as the web server. The value of this parameter must conform to the syntax specified in Section 1.1.1, "`uri` Parameter".

`time` Specifies the new value for the datalogger real time clock. This parameter is optional when the CSI Web Server is acting as the web server but is required when the datalogger is acting as the web server. If this parameter is not set, the CSI Web Server will use the host clock to determine the new datalogger time.

`format` Optionally specifies the expected format of the response. This can be one of `html` (the default when this parameter is not specified), `xml`, or `json`.

2.4.4. `ClockSet` Response

The response to the `ClockSet` command can be formatted as an HTML document, an XML document, or a JSON document depending upon the value of the `format` parameter. Regardless of the format, all responses will have the following parameters:

`outcome` Specifies a code that represents the outcome of the command. The following values are defined:

1. The clock was set.
2. The connection to LoggerNet failed.
3. Invalid LoggerNet logon.
4. Blocked by LoggerNet security.
5. Communication with the datalogger failed.
6. Communication with the datalogger is disabled.
7. Blocked by datalogger security.
8. An invalid `uri` parameter was specified.
9. LoggerNet is busy with the station.

`time` Specifies the value of the datalogger real time clock before it was set.

`description` Specifies a text description of the `outcome` code.

2.4.4.1. HTML Response Format

A response is sent as an HTML document when the value of the `format` parameter is set to `html` or that parameter is not specified. The output will be structured in an HTML table where each response parameter is on its own row. The following example demonstrates this format:

```
<html>

<head>
<title></title>
</head>
<body>
  <table border="1">
    <tr>
      <td>outcome</td><td>1</td>
    </tr>
    <tr>
      <td>time</td><td>2014-11-26 14:21:43.28</td>
    </tr>
    <tr>
      <td>description</td><td>The clock was set</td>
    </tr>
  </table>
</body>
</html>
```

2.4.4.2. XML Response Format

A response is sent as an XML document when the `format` parameter is set to `xml`. This document has a root element named `ClockSetResponse` and each response parameter is formatted as an attribute. The following example demonstrates this format:

```
<?xml version="1.0" encoding="UTF-8"?>
<ClockSetResponse description="The clock was set"
                  outcome="1"
                  time="2014-11-26T14:23:11.79" />
```

2.4.4.3. JSON Response Format

A response is sent as a JSON document when the `format` parameter is set to `json`. This document is formatted as a JSON object with the response parameters being represented as members to this object. The following example demonstrates this format:

```
{
  "outcome": 1,
  "time": "2014-11-26T14:27:32.73",
  "description": "The clock was set"
}
```

2.5. Files Related Commands

2.5.1. NewestFile Command

Both LoggerNet and dataloggers are able to manage collections of accumulated files, such as camera images. These files might have been stored by the datalogger on its USR: or CRD: drives or they might have been retrieved by LoggerNet. As these file accumulate, it is difficult for the client to keep track the file names. The NewestFile command can be used to retrieve the newest in a series of files using a wild card expression. This command requires a minimum access level of read-only. If the request authorisation maps to a lower access level, the server must respond with a 401 Authorization Required HTTP response.

The NewestFile command recognises the following parameters:

command — Must be specified as NewestFile.

expr — Specifies the complete path and wild card expression for the desired set of files. An appropriate value for the datalogger would be CRD:*.jpg which would pick the newest file on the card device with the JPG extension. If the web server is the CSI Web Server, the directory specified must be in the set allowed in the site configuration file (.sources.xml). This policy exists as a security measure in order to prevent the remote client from being able to access any file system on the host machine.

uri — Optionally specifies a data source URI that identifies a datalogger or HTTP data source. This parameter is ignored by when the datalogger is acting as the web server. If this parameter is omitted or is specified as an empty string, the expr will be applied on the host computer's file system. This value must conform to the syntax specified in Section 1.1.1, “uri Parameter”.

2.5.2. NewestFile Response

If a matching file is found, the web server will transmit the contents of that file as the content of the response. The web server should set the Content Type parameter of the HTTP response header to match the MIME type associated with the file extension. It should also set the Content-Disposition parameter in the HTTP response header to match the name of the file that was selected. If no matching file is found, the web server must send a 404 Not Found HTTP response.

2.5.3. ListFiles Command

The ListFiles command provides the client with the means of obtaining a list of file names and other meta-data in or below the web server's working directory. It can also be used with the CSI Web Server to get a list of files on the file system of a datalogger accessed through a LoggerNet source. This command requires an access level of read-only or higher. If the client's access level is lower than this, the web server must respond with a 40 Authorization Required HTTP response. The path specified in the HTTP request must specify the directory to be enumerated. The ListFiles command recognises the following parameters:

`command` Must be specified as `ListFiles`

`format` Optionally specifies the expected format of the response. Must be one of `html` (the default if this parameter is not specified), `xml`, or `json`.

`uri` Optionally specifies the data source URI for a station accessed through a LoggerNet data source. If this parameter is not specified, the web server will return the list from its own file system.

2.5.4. `ListFiles` Response

Depending upon the value of the `format` parameter of the `ListFiles` command, the web server will respond with the files list formatted as an HTML document, and XML document, or as a JSON document. Regardless of the format, the following parameters will be described for each file or directory:

`path` Specifies the path of the file or directory relative to the URL path.

`is_dir` Specifies a boolean flag that is set to a value of true if the path references a directory.

`size` Specifies an integer that gives the size of a file in bytes or the free space if the entry is for a directory.

`last_write` Specifies the date and time when the file or directory was last written or updated.

`run_now` Specifies a boolean flag that indicates whether the file is a datalogger marked to be the current datalogger program. The CSI Web Server will set this value to false when listing files from its own directory.

`run_on_power_up` Specifies a boolean flag that indicates whether the file is a datalogger program marked as the program to run on power up. The CSI Web Server will set this value to false when listing files from its own directory.

`read-only` Specifies a boolean flag that indicates that the file is read-only.

`paused` Specifies a boolean flag that indicates whether the file is a datalogger program that is in a paused state. The CSI Web Server will set this value to false when listing files from its own directory.

The web server should not report any file, such as `.sources.xml` and `.csipasswd` that begins with a period. These files should be considered to be protected and should not be returned if the client attempts to retrieve them either.

2.5.4.1. HTML Response Format

The response will be formatted as an HTML document when the value of the `format` request parameter is set to `html` or when that parameter is not specified. The document will be structured

as an HTML table where each row of the table represents a file or directory. The following is an example of this format:

```
<html>

<head>
<title>ListFiles Response</title>
</head>

<body>

<h1>
ListFiles Response
</h1>

<table border="1">

<tr>
<td>
<b>
Path
</b>
</td><td>
<b>
Is Directory
</b>
</td><td>
<b>
Size
</b>
</td><td>
<b>
Last Write
</b>
</td><td>
<b>
Run Now
</b>
</td><td>
<b>
Run On Power Up
</b>
</td><td>
<b>
Read Only
</b>
</td><td>
```

```
<b>
Paused
</b>
</td>
</tr>

<tr>
<td>CPU:</td>
<td>true</td>
<td>80384</td>
<td>2014-12-01 10:35:30.325</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>

<tr>
<td>CPU:lights-web.cr1</td>
<td>false</td>
<td>17921</td>
<td>2014-09-16 09:36:04</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
</table>

</body>

</html>
```

2.5.4.2. XML Response Format

The response will be formatted as an XML document when the value of the `format` request parameter is set to `xml`. The root element of this document will be named `ListFilesResponse` and each child element will describe a file or directory. The following is an example of this format:

```


<ListFilesResponse>
  <file is_dir="true"
        last_write="2014-12-01T10:40:37.444"
        path="CPU:"
```

```
        paused="false"
        read_only="false"
        run_now="false"
        run_on_power_up="false"
        size="80384" />
  <file is_dir="false"
        last_write="2014-09-16T09:36:04"
        path="CPU:lights-web.cr1"
        paused="false"
        read_only="false"
        run_now="true"
        run_on_power_up="true"
        size="17921" />
</ListFilesResponse>
```

2.5.4.3. JSON Response Format

The response will be formatted as a JSON document when the value of the `format` parameter is set to a value of `json`. The document will be structured as an object containing an array named `files`. Each element of that array will be an object that describes the file or directory. The following is an example of this format:

```
{
  "files": [
    {
      "is_dir": true,
      "name": "CPU:",
      "size": 80384,
      "last_write": "2014-12-01T10:45:05.038",
      "run_now": false,
      "run_on_power_up": false,
      "read_only": false,
      "paused": false
    },
    {
      "is_dir": false,
      "name": "CPU:lights-web.cr1",
      "size": 17921,
      "last_write": "2014-09-16T09:36:04",
      "run_now": true,
      "run_on_power_up": true,
      "read_only": false,
      "paused": false
    }
  ]
```

```
}
```

2.5.5. Sending a File using PUT

An HTTP client can send a file to the web server using the HTTP `PUT` method. The name and path of the file to be transmitted must be specified in the URL path. In order for the request to succeed, the request must authorise with an access level of `all`. If the client access level is less than this, the server must respond with a `401 Authorization Required` HTTP response.

2.5.6. SendFile Command

The `SendFile` command is used to request that the web server send the file with contents in the message body to the datalogger identified by the `uri` parameter. This command is supported only in the CSI web serve version 1.05.00.04 and newer. In order for this command to succeed, the client must have an access level of `all`. If the client's assigned access level is lower, then the server must respond with a `401 Authorization Required` HTTP response.

The only data source types that the `SendFile` command can be expected to work with are the LoggerNet data source and the HTTP data source.

The `SendFile` command requires the following parameters:

`command` The `command` parameter must be set to `SendFile`.

`uri` Specifies the data source URI that identifies the datalogger to which the file will be sent. For an HTTP data source, this will be the name of the data source. For LogggerNet data sources, this must include the data source name and the LoggerNet station name.

`path` Specifies the device and file name used by the datalogger to store the file. This parameter must conform to the following syntax:

```
path      := drive ":" file-name.
drive     := "CPU" | "USR" | "CRD" | "USB".
file-name := string.
```

`format` Optionally specifies the format of the server response. If this parameter is not specified, a value of `html` will be assumed by the server. Supported values include `html`, `json`, and `xml`.

2.5.7. SendFile Response

Depending upon the value of the `format` parameter of the `SendFile` command, the web server will format the response as an HTML document, a JSON document, or as an XML document with appropriate `Content-Type` header values. Regardless of the format, the following parameters will be given in the response:

`outcome` Specifies a code that represents the outcome of the command. The following values are defined:

- 0 An unrecognised error condition occurred while sending the file.
- 1 The file was successfully sent.
- 2 The connection to the data source failed.
- 3 Invalid logon parameters were specified for the data source.
- 4 The `uri` parameter did not identify a station that supports the send file operation.
- 5 Blocked by LoggerNet server security.
- 6 Communication with the station failed while sending the file.
- 7 Communication to the station is disabled.
- 8 An invalid file name was specified in the command `path` parameter.
- 9 The datalogger does not have resources available to store the file.
- 10 Blocked by datalogger security.
- 11 The datalogger drive root directory is full.
- 12 An invalid `uri` was specified or a `uri` was specified to a data source that does not support the file send operation.

`description` Specifies a text description of the `outcome` parameter.

2.5.8. `FileControl` Command

The `FileControl` command allows the client to perform certain operations on files on a datalogger. In order for this command to succeed, the client must have an access level of `all`. If the client has a lower access level, the server must respond with `401 Authorization Required` HTTP request. The `FileControl` command recognises the following parameters:

`command` The `command` parameter must be set to `FileControl`.

`uri` Specifies a data source URI that identifies the station on which the file control operation should take place. This parameter is required by the CSI web server (version 1.05 and newer) and must be associated with either an HTTP or a LoggerNet data source.

`action` Specifies the file control action that should take place. The following values are supported:

1. Compile and run the program specified by `file` and mark it as the program to run on power-up.
2. Mark the program specified by `file` as the program to run on power-up.
3. Mark the file specified by `file` as hidden.
4. Delete the file specified by `file`.
5. Format the device specified by `file`.
6. Compile and run the program specified by `file` and preserve existing data if possible.
7. Stop the currently running program.
8. Stop the currently running program and delete any associated data files.
9. Perform a full memory reset.
10. Compile and run the program specified by `file` but do not change the run on power-up program.
11. Pause the execution of the currently running program.
12. Resume the execution of the currently running program.
13. Stop the currently running program, delete any associated data, compile and run the program specified by `file` and mark it as the program to run on power up.
14. Stop the currently running program, delete any associated data, compile and run the program specified by `file` without affecting the program to be run on power-up.
15. Move the file specified by `file2` to the name and location specified by `file`.

16. Move the file specified by `file2` to the name and location specified by `file`, stop the currently running program, delete its associated data, and compile and run the program specified by `file` while marking it to run on power-up.

17. Move the file specified by `file2` to the name and location specified by `file`, stop the currently running program, delete its associated data, and compile and run the program specified by `file` without changing the program to run on power up.

18. Copy the file specified by `file2` to the name and location specified by `file`.

19. Copy the file specified by `file2` to the name and location specified by `file`, compile and run the program specified by `file`, and set it to run on power-up.

20. Copy the file specified by `file2` to the name and location specified by `file`, stop the currently running program, delete any of its associated data, and compile and run the program specified by `file` without affecting the program that will run on power-up.

`file` Specifies the first parameter for the file control operation. This parameter must be specified when the value of `action` is set to 1, 2, 3, 4, 5, 6, 10, 13, 14, 15, 16, 17, 18, 19, and 20.

`file2` Specifies the second parameter for the file control operation. This parameter must be specified when the value of `action` is equal to 15, 16, 17, 18, 19, and 20.

`format` Optionally specifies the expected format of the response. This can be one of `html` (the default if this parameter is not specified), `xml`, and `json`.

2.5.9. `FileControl` Response

The format of the response to the `FileControl` command depends upon the optional `format` parameter in the HTTP request. The format can either be formatted as an HTML table with each response parameter on its own row, an XML document with a root name of `FileControlResponse` and the response parameters specified as attributes, or as a JSON document with each response parameter formatted as its own member.

If the datalogger must reset as a result of the file control operation that was specified, it must transmit the complete HTTP response before doing so.

The following parameters must be included in the response:

`outcome` Specifies the outcome of the operation. The following values are defined:

- -1 An unrecognised error condition was identified.
- 0 The operation was completed successfully.
- 1 Permission denied by the datalogger
- 2 Invalid logon parameters specified.

3	The connection to the datalogger failed.
4	An invalid station URI was specified.
5	File Control is not supported.
6	Permission denied by the LoggerNet server.
7	Communication with the datalogger failed.
8	Communication with the datalogger is disabled.
9	Insufficient resources on the datalogger.
10	The LoggerNet server has the datalogger locked.
11	The datalogger root directory is full.
12	The datalogger is busy with the file.
13	An invalid value was specified for `file` or `file2`
14	The datalogger drive is busy.
19	An unsupported value was specified for `action`.
20	The file system directory is full.

`holdoff` Specifies an interval in seconds for which the web client should not attempt to communicate with the datalogger. If a value of zero is specified, communication can resume immediately. This parameter is needed because the datalogger will reset for many of the file control actions and it can be unresponsive for possibly tens of seconds.

`description` Specifies a text description of the code specified by `outcome`.

2.6. Alarms Related Commands

Starting with CSI Web Server version 1.02, it is now possible to publish projects that contains alarm. These alarms are described to the web server in the `.sources.xml` file read by the web server which also describes the data sources and allowed directories for that project. These alarms can be used to generate alerts via e-mail, send files via FTP, forward derived variable values to other data sources, or to simply flag exceptional conditions to web clients. When alarms are published to the CSI Web Server, that server becomes responsible for monitoring the data, apply any logic, and carrying out any actions associated with an alarm condition. The web server provides commands that allow the client to monitor the state of published alarms and, optionally acknowledge these alarms.

The commands described in this section are not supported by datalogger web servers.

2.6.1. `CheckAlarm` Command

The `CheckAlarm` command can be used to check the status of one of the alarms published to the web server. It can also be used to optionally acknowledge a triggered alarm. The command requires a minimum access level of `read-only` but will require a minimum access level of `read/write` when the `acknowledge` parameter is set to true. If the request specifies less access than the required level, the web server must respond with a `401 Authorization Required` HTTP response.

The `CheckAlarm` command recognises the following parameters:

`command`	Must be set to a value of `CheckAlarm`.
`name`	Specifies the name or unique ID for the alarm. If an alarm name is used, there is a chance that multiple alarms might use the same name. In that case, the alarm affected will be the first declared with that name.
`format`	Optionally specifies the expected format of the response. This parameter can be specified as `html` (the default if this parameter is not specified), `xml`, or `json`.
`acknowledge`	Optionally specifies whether the alarm should be acknowledged if it is in a signaled state. If this parameter is not specified or is set to false, the state of the alarm will not be changed.
`acknowledge_comments`	Optionally specifies the comments that will be logged when the alarm is acknowledged. This is generally supplied by the user. This parameter will be ignored unless the value of `acknowledge` is set to true.

2.6.2. `CheckAlarm` Response

If the client specifies an identifier or name for an alarm that is not defined for the web server, it will respond with an `404 Not Found` HTTP response. Otherwise, the format of the response will depend upon the value of the `format` command parameter.

Regardless of the format specified, the response will contain the following parameters:

`name` Specifies the name (user assigned) for the alarm.

`id` Specifies the unique generated identifier for this alarm. Typically, this is a GUID that gets generated by RTMC when the alarm is created.

`value` Specifies the value that was last evaluated by the alarm's source expression.

`value_type` Specifies a data type for the `value` parameter. This can be one of `xsd:string`, `xsd:double`, or `xsd:boolean`.

`state` Specifies the current state of the alarm. This value can be one of the following:

- `on` One of the alarm's conditions has been asserted and the alarm has not been acknowledged by any client.
- `off` None of the alarm's conditions are triggered and there is no need for acknowledgement.
- `acknowledged` One of the alarm's conditions is still asserted but the alarm has been acknowledged.

`last_error` Specifies the last error that the alarm reported for its source expression. This value will be an empty string if there is no error condition.

`triggered_condition_name` Specifies the name of the alarm's currently triggered condition. This value will be empty if the alarm is not triggered.

`actions_pending` Specifies the number of actions that are currently pending for this alarm.

`last_action_error` Specifies an error string that describes any error for the last action. If the last action succeeded, this value will be an empty string.

2.6.2.1. HTML Response Format

If the value of `format` is not specified or is specified as `html`, the response will be formatted as an HTML document which contains a table for which each row specifies a response parameter. The following shows an example of this response:

```
<html>

<head>
<title></title>
</head>

<body>

<h1>
CheckAlarmResponse
</h1>

<table border="1">

<tr>
<td>
<tt>
name
</tt>
</td><td>Basic Alarm</td>
</tr>

<tr>
<td>
<tt>
id
</tt>
</td><td>650f7b8d-b670-4433-8ccc-2b7b95882f0c</td>
</tr>

<tr>
<td>
<tt>
value
</tt>
</td><td>73.55914</td>
</tr>

<tr>
<td><tt>value_type</tt></td>
<td>xsd:double</td>
</tr>

<tr>
<td>
<tt>
state
</tt>
</td><td>off</td>
```

```
</tr>

<tr>
<td>
<tt>
last_error
</tt>
</td>
<td>
</td>
</tr>

<tr>
<td>
<tt>
triggered_condition_name
</tt>
</td><td></td>
</tr>

<tr>
<td>
<tt>
actions_pending
</tt>
</td><td>0</td>
</tr>

<tr>
<td>
<tt>
last_action_error
</tt>
</td><td></td>
</tr>

<tr>

</tr>

</table>

</body>

</html>
```

2.6.2.2. XML Response Format

The response to the `CheckAlarm` command will be formatted as an XML document when the value of `format` is set to `xml`. The root element if this document will be named `CheckAlarmResponse` and each of the response parameters will be represented as a child element. The following is an example of this format:

```
{
  "name": "Basic Alarm",
  "id": "650f7b8d-b670-4433-8ccc-2b7b95882f0c",
  "value": "73.55914",
  "value_type": "xsd:double",
  "state": "off",
  "last_error": "",
  "triggered_condition_name": "",
  "actions_pending": 0,
  "last_action_error": ""
}
```

2.6.3. `ListAlarms` Command

The `ListAlarms` command can be used to get obtain a list of all alarms defined for a site. This command does not change the state of any of these alarms. The minimum access level required for this command is `read-only`. If a client does not have this access, the web server must respond with a `401 Autorization Required` HTTP response.

The `ListAlarms` command recognises the following parameters:

`command` This value must be set to `ListAlarms`.

`format` Optionally specifies the format of the response. Can be one of `html` (the default if this parameter is not specified), `xml`, and `json`.

2.6.4. `ListAlarms` Response

The format of the `ListAlarms` response will depend upon the value of the `format` parameter specified in the command. The parameters specified for each alarm will correspond with those in the `CheckAlarm` response (see Section 2.6.2, "`CheckAlarm` Response"). Regardless of the format, each alarm that is defined for the site will have the following parameters:

`name` Specifies the user assigned name for the alarm. This value can be used to identify an alarm in the `CheckAlarm` command (see Section 2.6.1, "`CheckAlarm` Command") but is not guaranteed to be unique.

`id` — Specifies a unique identifier for this alarm. This value is typically specified as a GUID.

`value` — Specifies the current value of the alarm source expression.

`value_type` — Specifies the data type for the `value` parameter. This value can be one of `xsd:double`, `xsd:string`, or `xsd:boolean`.

`state` — Specifies the current state of the alarm. This value can be one of the following:

- `on` — One of the alarm's conditions has been asserted and the alarm has not been acknowledged by any client.
- `off` — None of the alarm's conditions are triggered and there is no need for acknowledgement.
- `acknowledged` — One of the alarm's conditions is still asserted but the alarm has been acknowledged.

`last_error` — Specifies the last error that the alarm reported for its source expression. This value will be an empty string if there is no error condition.

`triggered_condition_name` — Specifies the name of the alarm's currently triggered condition. This value will be empty if the alarm is not triggered.

`actions_pending` — Specifies the number of actions that are currently pending for this alarm.

`last_action_error` — Specifies an error string that describes any error for the last action. If the last action succeeded, this value will be an empty string.

2.6.4.1. HTML Response Format

The response to the `ListAlarms` command will be formatted when the value of the `format` parameter is set to `html` or when that value is not present. The HTML document will be formatted as a table with each alarm occupying one row and each alarm parameter occupying one column. The following example demonstrates this format:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
```

```
<title></title>
</head>

<body>

<h1>
ListAlarmsResponse
</h1>

<table border="1">

<tr>
<th>
<b>
name
</b>
</th><th>
<b>
id
</b>
</th><th>
<b>
value
</b>
</th>
<th><b>value_type</b></th>
<th>
<b>
state
</b>
</th>
<th><b>last_error</b></th>
<th>
<b>
triggered_condition_name
</b>
</th><th>
<b>
actions_pending
</b>
</th><th>
<b>
last_action_error
</b>
</th>
</tr>

<tr>
<td>Basic Alarm</td><td>ef579b87-66b1-4d95-82b7-aca1a3e32d9e</td>
```

```
<td>73.55914</td>
<td>xsd:double</td>
<td>off</td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>

</table>

</body>

</html>
```

2.6.4.2. XML Response Format

The response is formatted as an XML document when the value of the `format` parameter is set to `xml`. The root element of this document is named `ListAlarmsResponse`. Each of the defined alarms is represented by an `alarm` child element and the alarm attributes will be represented as child elements to the `alarm` element. The following example demonstrates this format:

```
<ListAlarmsResponse>
  <alarm>
    <name>Basic Alarm</name>
    <id>ef579b87-66b1-4d95-82b7-aca1a3e32d9e</id>
    <value>73.55914</value>
    <value_type>xsd:double</value_type>
    <state>off</state>
    <last_error/>
    <triggered_condition_name/>
    <actions_pending>0</actions_pending>
    <last_action_error/>
  </alarm>
</ListAlarmsResponse>
```

2.6.4.3. JSON Format Response

The response is formatted as a JSON document when the value of the `format` parameter is set to `json`. This document is structured as an object that contains a single array property called `alarms`. Each defined alarm is listed as an object in this array. The following example demonstrates this format:

```
{
  "alarms": [
    {
      "name": "Basic Alarm",
      "id": "ef579b87-66b1-4d95-82b7-aca1a3e32d9e",
      "value": "73.55914",
      "state": "off",
      "last_error": "",
      "value_type": "xsd:double",
      "triggered_condition_name": "",
      "actions_pending": 0,
      "last_action_error": ""
    }
  ]
}
```

2.7. Commands Associated with Access Tokens

This section describes commands that can be used by the client to obtain an access token that can be used with `Bearer` type authorisation instead of having to provide the user name and password for `Basic` authorisation.

2.7.1. `GetAccessToken` Command

This command can be used by the client to request that the server generate a pair of access tokens that can be used to authorise HTTP requests.

The `GetAccessToken` command recognises the following parameters:

`command` Must be set to a value of `GetAccessToken`.

`scope` Optionally specifies the scope of the requested tokens. If specified with a value of `off_line`, the server will return a refresh token with no set expiration date that can be used at any time to get a new set of access tokens. This feature is useful for clients that shouldn't store user credentials but need periodic access to server resources.

This command must be sent using the `POST` method and must also specify a `Content-Type` of `application/json`. The content must be a JSON formatted structure that contains the following properties:

`grant_type` (string) Must specify a value of `password`

`credentials` (object) Must specify an object with the following properties:

- `username` (string) Specifies the name of the account to use for generating the access token.
- `password` (string) Specifies the password associated with the user account.

2.7.2. `RefreshAccessToken` Command

This command can be used by the client to obtain a new set of access tokens from the server using a refresh token that was returned by an earlier request. This command recognises the following parameters:

`command` Must be set to a value of `RefreshAccessToken`

The request must be sent using HTTP POST and the request content must be JSON document that has the following properties:

`refresh_token` (string) Specifies the refresh token that was obtained using a previous `GetAccessToken` or `RefreshAccessToken` request.

2.7.3. `GetAccessToken` and `RefreshAccessToken` Response

The server must respond to the `GetAccessToken` or `RefreshAccessToken` command with a JSON document in its content. If the attempt to get the access token fails, the server will respond with an HTTP error (500) and the response body will be a JSON document that describes the circumstance of the failure. A successful response must have the following properties:

`access_token` (string)	Specifies the access token that the client must use to authorise future requests.
`expires_in` (number)	Specifies the interval in seconds during which the server will accept the access token between the time that it was issued and the time that it will expire.
`token_type` (string)	Must be set to a value of `Bearer`
`access_level` (number)	Specifies the access level assigned to the account at the time that the access token was generated. This can be used by the client to determine what parts of the UI should be disabled based upon access.
`refresh_expires_in` (number)	Specifies the interval in seconds from the time that the refresh token was generated for which the server will accept the refresh token. If set to a value of zero, the generated refresh token is an off-line token and can be used at any time to refresh both tokens.
`refresh_token` (string)	Specifies the token that must be passed with the `RefreshAccessToken` (see Section 2.7.2, "`RefreshAccessToken` Command") in order to obtain new access and refresh tokens.

3. Using Web Sockets

3.1. Introduction to Web Sockets

Starting with version 1.04 of the CSI Web Server, it is possible for the client to monitor data and alarms using Web Sockets. Web Sockets is a protocol related to HTTP that allows a web client and server to asynchronously exchange messages on a stream. The definition of the Web Sockets protocol can be found at http://tools.ietf.org/html/rfc6455. It starts out with an HTTP request with an upgrade specification. Once the server generates an appropriate response, the client and the server can send binary or text messages at any time. The content of these messages is up to the "sub-protocol" that is negotiated between the server and the client during the HTTP upgrade.

The web socket starts when the client sends an HTTP request similar to this example:

```
GET /engsoft HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: com.campbellsci.webdata
Sec-WebSocket-Version: 13
```

The web server must respond to this request appropriately as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: com.campbellsci.webdata
```

3.2. Web Sockets Implementation Details

In order to fit in with the other services in the datalogger web API, use of Web Sockets will have the following features:

- Older web servers will assume that all command messages received on a web socket will have read-only access requirements. Newer web servers will assign an access level to the web socket at the time of its creation based upon the access level granted to the `Authorization`

field of the upgrade header. A client can change that authorisation in these newer versions by sending a `logon` command message (see Section 3.3.7, “`Logon` Command”).

- The path specified in the request URI will specify the path to a published site.
- All messages passed to or from the server will be text and will be formatted as JSON documents.

3.3. Messages Passed Over Web Sockets

The following messages can be sent on an established web socket:

3.3.1. AddRequests Command

The AddRequests command is sent from the client to the server to add one or more data source requests for that client. The message is formatted as a JSON document that is structured as an object that contains an array of request objects. The parameters of these request objects are similar to those specified by the DataQuery command (see Section 2.1.1, "DataQuery Command").

Any requests specified in this message will be appended to requests that have already been started. The client must provide a unique transaction parameter for each new request. The web server will track all of the requests that have been added by the client and report any data using the RequestRecords notification message (see Section 3.3.3, "RequestRecords Notification"). These requests must maintain until one of the following happens:

- The client closes the request using the RemoveRequests message (see Section 3.3.5, "RemoveRequests Command").
- The request fails. The server must notify the client of this using the RequestFailed notification message (see Section 3.3.4, "RequestFailed Notification").
- The web socket is closed or broken.

The AddRequests message recognises the following parameters:

message — This member must be set as a string to AddRequest

requests — Specifies an array that contains one or more data source request objects. Each request will be a JSON object that contains the following fields:

- uri — Specifies the data source URI for the data to be monitored. This must reference either a table or a field in a table. This value must conform to the syntax described in Section 1.1.1, "uri Parameter".
- mode — Specifies the nature of the data query. The following values are supported:
  - most-recent — Specifies that up to the number specified by p1 of most recent records logged in the table should be returned.
  - since-time — Specifies all of the records logged since the time stamp specified in p1 should be returned.
  - since-record — Specifies that all records with a record number greater than or equal to the value specified in p1 should be returned. If that record number is not present in the table, data should returned starting with the oldest record in the table.

`date-range` Specifies that all data contained in the half-open time interval specified by the `p1` and `p2` parameters which are interpreted as time stamps. Records will be returned that have a time stamp greater than or equal to the time stamp specified by `p1` and less than the time stamp specified by `p2`.

`backfill` Specifies that all records starting with the first record that has a time stamp greater than or equal to the time stamp of the newest record less the time interval specified by `p1` in seconds.

`transaction` Specifies a transaction number that will uniquely identify this request in messages sent on this web socket connection.

`p1` Specifies qualifying information based upon the value of the `mode` parameter. The interpretation of this parameter varies as follows:

`most-recent` Specifies the maximum number of records to return.

`since-record` Specifies the record number at which the server should start sending records.

`since-time`, `date-range` Specifies the starting record time stamp. The first record that has a time stamp that is greater than or equal to this value will be selected as the starting point.

`backfill` Specifies a time interval in units of seconds that should be subtracted from the time stamp of the newest table record in order to find the starting record.

`p2` Specifies further qualifying information for some values of the `mode` parameter. The interpretation of this parameter varies as follows:

`date-range` Specifies the ending time stamp of the half-open time interval. The records returned will have a time stamp less than this value.

`since-record` Optionally specifies a starting time stamp to be used with data sources, such as

data files and databases, that have the possibility of having duplicate record numbers.

All others — This parameter will be ignored for any other value of `mode`.

`order` — Specifies the order in which the records should be reported in the response from the CSI Web Server. This parameter is ignored by the datalogger. The following values are defined:

- `real-time` — Specifies that the server will skip over historic records in favour of most recently logged records.
- `collected` — Specifies that the records should be returned in the order in which they were collected by LoggerNet. This order can differ from logged order when one-way or data advise data collection is used in conjunction with hole collection.
- `logged-with-holes` — Specifies that the data should be reported in the order in which it was logged by the datalogger. If there are holes that have not yet been collected by LoggerNet, newer data will not be returned until those holes have been collected or they are deemed no longer collectible by LoggerNet.
- `logged-without-holes` — Specifies that the data should be reported in the order in which it was logged by the datalogger. If there are uncollected holes, those records will be skipped in favour of more recent records.

`refresh` — Optionally specifies the mininum interval, in seconds, at which the web server should poll the table specified by the `uri`

parameter. This parameter will be ignored by for any data source type except LoggerNet sources.

For each request in the set specified in the `AddRequests` message, the server will send either a `RequestStarted` (see Section 3.3.2, "`RequestStarted` Notification") or a `RequestFailed` (see Section 3.3.4, "`RequestFailed` Notification") message.

3.3.1.1. `AddRequests` Example

The following shows an example of the `AddRequests` message format where a single request backfilled by an interval of one day is being added:

```
{
  "message": "AddRequests",
  "requests": [
    {
      "uri": "localhost:cr000.one_hour.temp_degf_avg",
      "mode": "backfill",
      "transaction": 1,
      "p1": 86400,
      "order": "collected"
    }
  ]
}
```

3.3.2. `RequestStarted` Notification

The `RequestStarted` notification is a message sent by the server to the client when a data source request has been reported as started. This message will be formatted as a JSON object that specifies the request transaction number as well as the CSIJson header for that request. The format of this message follows:

`message` Must be set to `RequestStarted`.

`transaction` Specifies the value of the `transaction` that was given in the original client request.

`head` Specifies the CSIJson header for the data that will be returned for the request. The complete header will be specified here while the truncated header will be provided with the `RequestRecords` notification (see Section 3.3.3, "`RequestRecords` Notification").

3.3.2.1. `RequestStarted` Example

An example of this format follows:

```
{
  "message": "RequestStarted",
  "transaction": 1,
  "head": {
    "transaction": 1,
    "signature": 64455,
    "environment": {
      "station_name": "cr1000",
      "table_name": "one_hour"
    },
    "fields": [
      {
        "name": "temp_degf_avg",
        "type": "xsd:float",
        "units": "DegF",
        "process": "Avg",
        "settable": false
      }
    ]
  }
```

3.3.3. RequestRecords Notification

The RequestRecords message is sent by the server to the client to report records for a request. The data will be reported using the CSIJson format. Since the complete header will have already been reported in the RequestStarted notification message, this format will contain the truncated header. The parameters sent for this message include the following:

message — Must be set to a value of RequestRecords

transaction — Specifies the request transaction number that was specified in the Section 3.3.1, "AddRequests Command" message.

records — Specifies the data to be reported formatted as a CSIJson object.

An example of this format follows:

```
{ "message": "RequestRecords",
  "transaction": 5,
  "records":{
    "head": {
      "transaction": 5,
      "signature": 44130
    },
    "data": [
      {
        "no": 23302,
        "time": "2015-08-24T07:10:00",
        "vals": [ 12.85, 32, 68.14, 7.131, 0, 0,
                  1725.3, 1729.3, "C", 4, "HZ",
                  "Haze" ]
      }
     ],
     "more": true
  }
}
```

3.3.4. `RequestFailed` Notification

The `RequestFailed` notification is a message sent by the server to the client when a data source request failure has been reported. The message will be formatted as a JSON object that specifies the request transaction number and any available information about the failure. The parameters for this message follow:

`transaction` Specifies the client assigned transaction that was specified in the `AddRequests` command message.

`failure` Specifies a numeric code that identifies the type of failure. This can be one of the following values:

0. An unrecognised failure condition has occurred.
1. An invalid data source name was specified.
2. The data source connection failed.
3. Invalid LoggerNet logon parameters.
4. An invalid station name was specified.
5. An invalid table name was specified.
6. Blocked by LoggerNet security.
7. An invalid `mode` or unsupported start option was specified.
8. An invalid order option was specified.
9. The table was deleted.
10. The station was shut down.
11. Unsupported operation was attempted.
12. An invalid column name was specified.
13. An invalid array address was specified.
14. An invalid `transaction` parameter was specified.

`description` Specifies a text description of the `failure` code.

3.3.5. RemoveRequests Command

The RemoveRequests command is sent from the client to the server to remove one or more requests that have already been started. The following parameters are recognised for this command:

message	Must be set to a value of RemoveRequests.
transactions	Specifies an array of transaction numbers. These number must match the values specified when the AddRequests command message was sent (see Section 3.3.1, "AddRequests Command").

3.3.5.1. RemoveRequests Example

```
{
  "message": "RemoveRequests",
  "transactions": [ 1 ]
}
```

3.3.6. AlarmChanged Notification

The AlarmChanged notification is sent by the server to the client to report that the state of an alarm has changed. One of these messages will be sent by the server when the web socket is first created and one message will be sent each time that the state of an alarm has changed. The message is formatted as a JSON object and will have the following parameters:

message	This value must be specified as AlarmChanged.
name	Specifies the user assigned name for the alarm. This value can be used to identify an alarm in the CheckAlarm command (see Section 2.6.1, "CheckAlarm Command") but is not guaranteed to be unique.
id	Specifies a unique identifier for this alarm. This value is typically specified as a GUID.
value	Specifies the current value of the alarm source expression.
value_type	Specifies the data type for the value parameter. This value can be one of xsd:double, xsd:string, or xsd:boolean.
state	Specifies the current state of the alarm. This value can be one of the following:

	`on`	One of the alarm's conditions has been asserted and the alarm has not been acknowledged by any client.
	`off`	None of the alarm's conditions are triggered and there is no need for acknowledgement.
	`acknowledged`	One of the alarm's conditions is still asserted but the alarm has been acknowledged.
`last_error`		Specifies the last error that the alarm reported for its source expression. This value will be an empty string if there is no error condition.
`triggered_condition_name`		Specifies the name of the alarm's currently triggered condition. This value will be empty if the alarm is not triggered.
`actions_pending`		Specifies the number of actions that are currently pending for this alarm.
`last_action_error`		Specifies an error string that describes any error for the last action. If the last action succeeded, this value will be an empty string.

3.3.6.1. `AlarmChanged` Example

The following fragment demonstrates the message format of the `AlarmChanged` message:

```
{
  "message": "AlarmChanged",
  "name": "Basic Alarm",
  "id": "650f7b8d-b670-4433-8ccc-2b7b95882f0c",
  "value": "73.55914",
  "value_type": "xsd:double",
  "state": "off",
  "last_error": "",
  "triggered_condition_name": "",
  "actions_pending": 0,
  "last_action_error": ""
}
```

3.3.7. Logon Command

Requests that a new authorisation level be given to this web socket session. The web server will assign the initial level for the session based upon the access level derived from the logon parameters given in the `Authorization` HTTP header field of the HTTP upgrade request that started the web socket.

The server must respond to this message with the `LogonAck` message (see Section 3.3.8, "`LogonAck` Message") message. This message is formatted as a JSON document that will have the following parameters:

`message` This value must be specified as `logon`.

`transaction` Specifies a token that must be repeated in the acknowledgement message to help the client identify the response.

`user_name` Specifies the user name associated with the account that should be used.

`password` Specifies the password associated with the account that should be used.

`access_token` Specifies the access token returned by the `GetAccessToken` and `RefreshAccessToken` commands (see Section 2.7, "Commands Associated with Access Tokens"). This value can be specified instead of the user name and password.

An example of this format follows:

```
{ "message": "logon",
  "transaction": 1,
  "user_name": "leonidas",
  "password": "sparta"
}
```

3.3.8. LogonAck Message

The server will send this message in response to the `Logon` command message (see Section 3.3.7, "`Logon` Command"). This message will report the access level that the server has assigned to the web socket session. The format of this message follows:

`message` This value must be specified as `LogonAck`.

`transaction` Specifies the `transaction` token that the client specified in the `Logon` message.

`access` Specifies the access level that is now assigned to the web socket session. This value must be one of the following:

0 Indicates that all access is denied. This can happen if the the client specified a non-existent account, the password was wrong, or an account was referenced that is disabled.

1 All access allowed

2 Read/write access allowed

3 Read-only access allowed

The following is an example of this message format:

```
{
   "message": "LogonAck",
   "transaction": 1,
   "access": 1
}
```

3.3.9. `StartTerminal` Command Message

This command message is sent by the client to request that a terminal session be started on the web socket. This message will provide a transaction token that will be used to identify all future terminal messages for this session. It will also identify the URI for the station with which terminal I/O should be conducted.

In order to perform terminal I/O with a station, a client must first send this message to request the resource from the server. If the web socket session has the requisite permission (likely full access) and the transaction parameters check out, the web server will respond with an initial `TerminalData` message (see Section 3.3.10, "`TerminalData` Message") that will indicate whether the session could begin. Thereafter, the client can send the `TerminalData` message when it has data to transmit and the server will also send the `TerminalData` message when it has data to transmit. Either side will be able to cancel the terminal session by either closing the host web socket or by sending a `TerminalData` message with the appropriate close code.

`message` Must be set to a value of `StartTerminal`.

`transaction` Specifies the transaction token that will identify all future terminal I/O with the specified station.

`station_uri` Specifies the data source URI that identifies the station with which terminal I/O will be conducted. This parameter will be ignored by a datalogger acting as a web server.

The following is an example of a `StartTerminal` command:

```
{
   "message": "StartTerminal",
   "transaction": 1,
   "station_uri": "ln:cr6"
}
```

3.3.10. `TerminalData` Message

This message can be sent by either the client or the server once a terminal session has been started with the `StartTerminal` command message (see Section 3.3.9, "`StartTerminal` Command Message"). It will convey any data that is to be moved to or from the terminal as well as the state of the terminal session. The following describes the format of this message:

`message` Must be set to a value of `TerminalData`.

`transaction` Specifies the `transaction` property that the client first specified in the `StartTerminal` message.

`status` Specifies the current status if the terminal session. This must be one of the following values:

1. The session is still active.
2. Sent by the server to indicate that the web socket session does not have permission to keep the session going.
3. Sent by the server when the client sends a `TerminalStart` command that uses a `transaction` token that is already in use on the web socket session.
4. Sent by the server to indicate that a terminal session is already in progress with the station specified in the `StartTerminal` command.
5. Sent by the server to indicate that the station does not exist or has been deleted.
6. Sent by the client or the server to indicate that the terminal session must end.

`binary` Optional property that can be set to true to indicate that the content of this message is binary and therefore encoded as base64. If this property is omitted, the receiver will assume that the content is UTF-8 encoded text.

`content` Specifies the data that is to be sent or has been received from the datalogger. This field can be empty if this message is sent to convey status data. This string will be interpreted as UTF-8 encoded unicode text if the `binary` property is set to false or omitted. If the `binary` property is set to true, this string will be decoded as BASE64 encoded binary.

The following example shows a `TerminalData` message received from a datalogger:

```
{
   "message": "TerminalData",
   "transaction": 1,
   "status": 1,
   "binary": false,
   "content": "\r\nCR6>"
}
```

A. Log of Changes to This Document

Version 1.00.11 - 22 April 2021 (Jon Trauntvein)

Added commands to obtain or refresh access tokens and added the ability to use access tokens as authorisation in API requests and also in the `logon` websocket command (see ???.

22 November 2017

- I have added a description of the `CheckAuthorization` (see Section 2.3.3, "`CheckAuthorization` Command") command.

20 April 2017 (Jon Trauntvein)

- I have added a style sheet asnd profile for compiling this document for customer consumption.

23 September 2016 (Jon Trauntvein)

- I have added new web socket messages to deal with logging in (see Section 3.3.7, "`Logon` Command" and Section 3.3.8, "`LogonAck` Message") as well as terminal services (see Section 3.3.9, "`StartTerminal` Command Message" and Section 3.3.10, "`TerminalData` Message").

31 May 2016 (Jon Trauntvein

- I have added new response codes the file control response (see Section 2.5.9, "`FileControl` Response").

12 May 2016 (Jon Trauntvein)

- I have added a new HTTP command to send files to the datalogger via the CSI web server (see Section 2.5.6, "`SendFile` Command").

- I have added a new optional parameter, `uri`, for the `FileControl` command (see Section 2.5.8, "`FileControl` Command").

26 August 2015 (Jon Trauntvein)

- I have corrected the web socket service that should be reported and requested from `com.campbellsci.logger-web` to `com.campbellsci.web-data`. See Section 3.1, "Introduction to Web Sockets".

- I have added a parameter that was missing in the web sockets `RequestRecords` notification (see Section 3.3.3, "`RequestRecords` Notification"). The `transaction` fields is now described.

- I have added an example of the `RequestRecords` web socket message (see Section 3.3.3, “`RequestRecords` Notification”).

8 December 2014 (Jon Trauntvein)

- I have added a section that writes about the proposed web sockets extensions.

2 December 2014 (Jon Trauntvein)

- I have rewritten the API document so that a PDF can be generated in order to make it easier to share.

- I have removed much of the text from the original document that dealt with proposals.