In Process REST Server for SoftPLC® Runtime

Version 1.6

Table of Contents

Chapter 1. Terms of Use

Because of the variety of uses of the information described in this manual, the users of, and those responsible for applying this information must satisfy themselves as to the acceptability of each application and use of the information. In no event will SoftPLC Corporation be responsible or liable for its use, nor for any infringements of patents or other rights of third parties which may result from its use.

SOFTPLC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE.

SoftPLC Corporation reserves the right to change product specifications at any time without notice. No part of this document may be reproduced by any means, nor translated, nor transmitted to any magnetic medium without the written consent of SoftPLC Corporation.

SoftPLC, and TOPDOC are registered trademarks of SoftPLC Corporation.

First Printing May, 2025

Latest Printing May, 2025

SoftPLC Corporation
25603 Red Brangus Drive
Spicewood, Texas 78669

USA Telephone: 1-800-SoftPLC
WW Telephone: 512/264-8390
URL: http://softplc.com
Email: support@softplc.com

Chapter 2. Overview

2.1. Introduction

The SoftPLC runtime is control software developed by SoftPLC Corporation. It is is embedded into all SoftPLC controllers and gateways. This document describes how to configure and use the WEB.TLM add-on module for the SoftPLC runtime. This module enables the controller to be a web server, and by result, provides a REST API server.

2.2. Concepts

The SoftPLC runtime engine software supports TLMs (TOPDOC Loadable Modules), which are extensions to SoftPLC. A TLM may be loaded either as a DRIVER or as a MODULE. The difference between a DRIVER and a MODULE is that a DRIVER is called once per SoftPLC scan cycle, and optionally an additional number of times per scan. A MODULE is only called when the control program decides to call it and not as an inherent part of the scan. TLMs are made known to SoftPLC in a MODULES.LST file which may be edited by TOPDOC NexGen by traversing to: **PLC › Modules**.

This document describes a server that is both a web server and a REST (API) server that is available for the SoftPLC runtime environment.

2.3. Features

- Web server functionality which can serve files and directory content.
- REST API server as described below.
- REST requests can be made using either datatable addresses or tagnames.
- The API supports both simple and scattered reads and writes. Scattered operations are done on non-contiguous datatable locations and can result in fewer transactions and thereby improved efficiency.
- This TLM, like all TLMs, runs in the same process as the SoftPLC runtime and therefore has direct memory access to the SoftPLC runtime datatable.
- Any client which can issue HTTP requests can interoperate with this server. That includes browser Javascript, Python, n8n, etc.

2.4. Limitations

- The total number of supported HTTP connections from clients is over 50.

2.5. Requirements

- SoftPLC version 5.1 or greater
- TOPDOC NexGen version 5.1 or greater

2.6. Terminology

The HTTP protocol establishes several request types, sometimes called verbs, which are used to read and write data between a client and server. REST servers are fairly consistent in their use of these request types. The request types are GET, POST, PUT. GET is used for reads. POST is similar to GET, except that the description of the desired data to be read can be larger and is contained in the body of the request rather than in the URL. PUT is used to write data to the server.

XML is a text format that is used in REST servers. include link to XML tutorial.

In our case, XML is used to format POST and PUT request bodies, and is also used in the response to a GET.

Chapter 3. Web TLM Configuration

3.1. Configuration Steps

1. **Module Installation** - Ensure the TLM is installed in the SoftPLC
2. **Enable TLM** – Configure SoftPLC to load the TLM at startup
3. **Configure TLM** – Edit the driver configuration file WEB.LST for your desired communications
4. **Configure other Device(s)** – Following vendor instructions, set up the other device(s) to communicate with the SoftPLC (not described in this document)

3.1.1. Module Installation

The TLM is a file named **web.tlm.so** and the configuration file is named **WEB.LST**. If purchased, these files are pre-installed on the SoftPLC CPU or Gateway in the /SoftPLC/tlm directory.

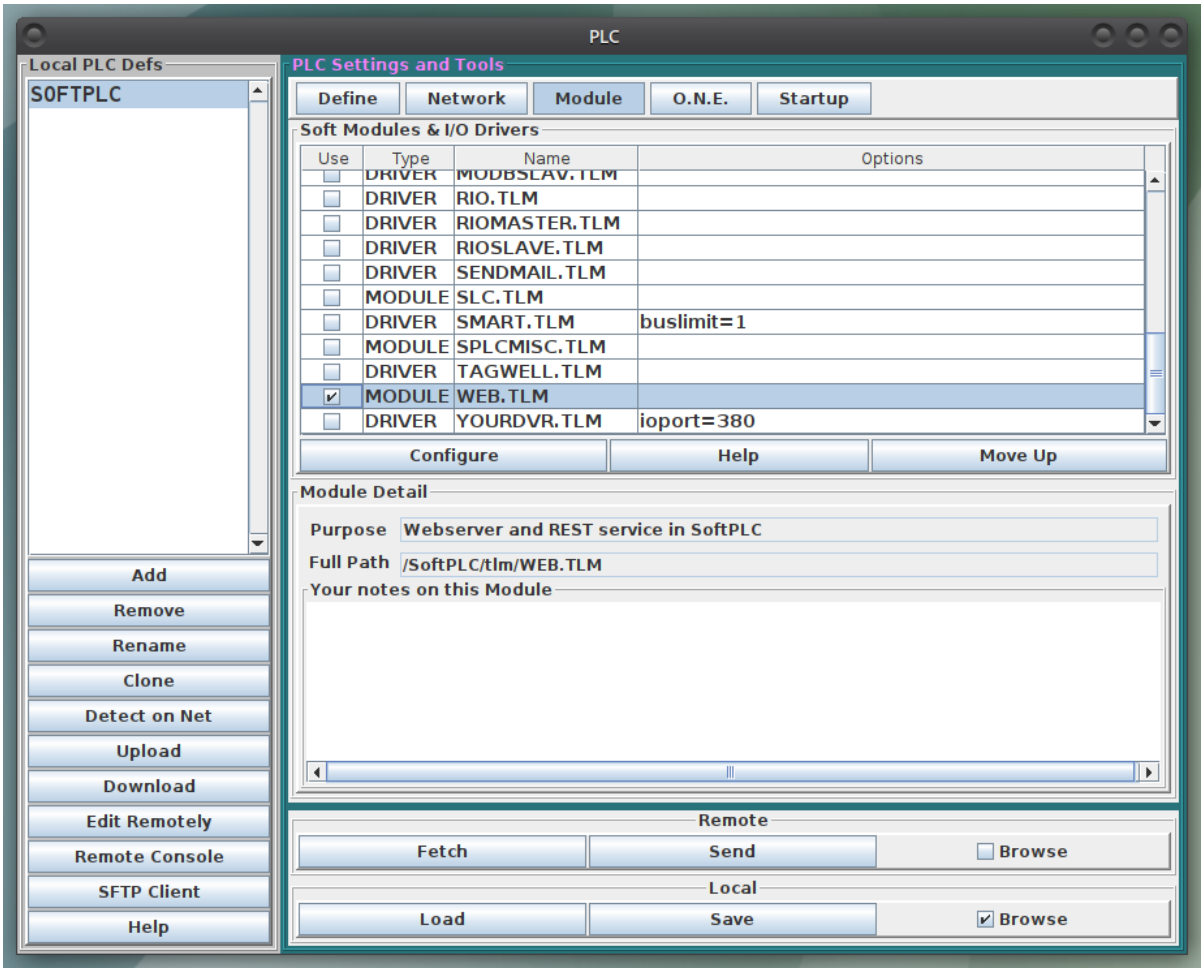
Web TLM Option

The Web TLM is an optional add-on at time of purchase. Please contact SoftPLC Support to verify installation (or procure a copy) of the Web TLM for a particular device. Please have serial number of the specific device at the ready. Field installation of the Web TLM is possible.

3.1.2. Enable Web TLM

The TOPDOC NexGen Manual and help system describe how to use the Module Editor to enable and configure TLM's. The following sections assume previous knowledge of TOPDOC NexGen's Module Editor, and other SoftPLC configuration procedures.

To enable the TLM, use TOPDOC NexGen to traverse to **PLC › Modules** and check **Use** for WEB.TLM. There are no **Options** for this TLM.



Save Enabled Modules

The **[Save]** button will write the MODULE.LST file to the development system's disk.
The **[Send]** button will write the MODULE.LST file to the runtime system's disk.

It is good practice to **both [Save]** and **[Send]** the edits, this way both your development system and the SoftPLC get a copy.

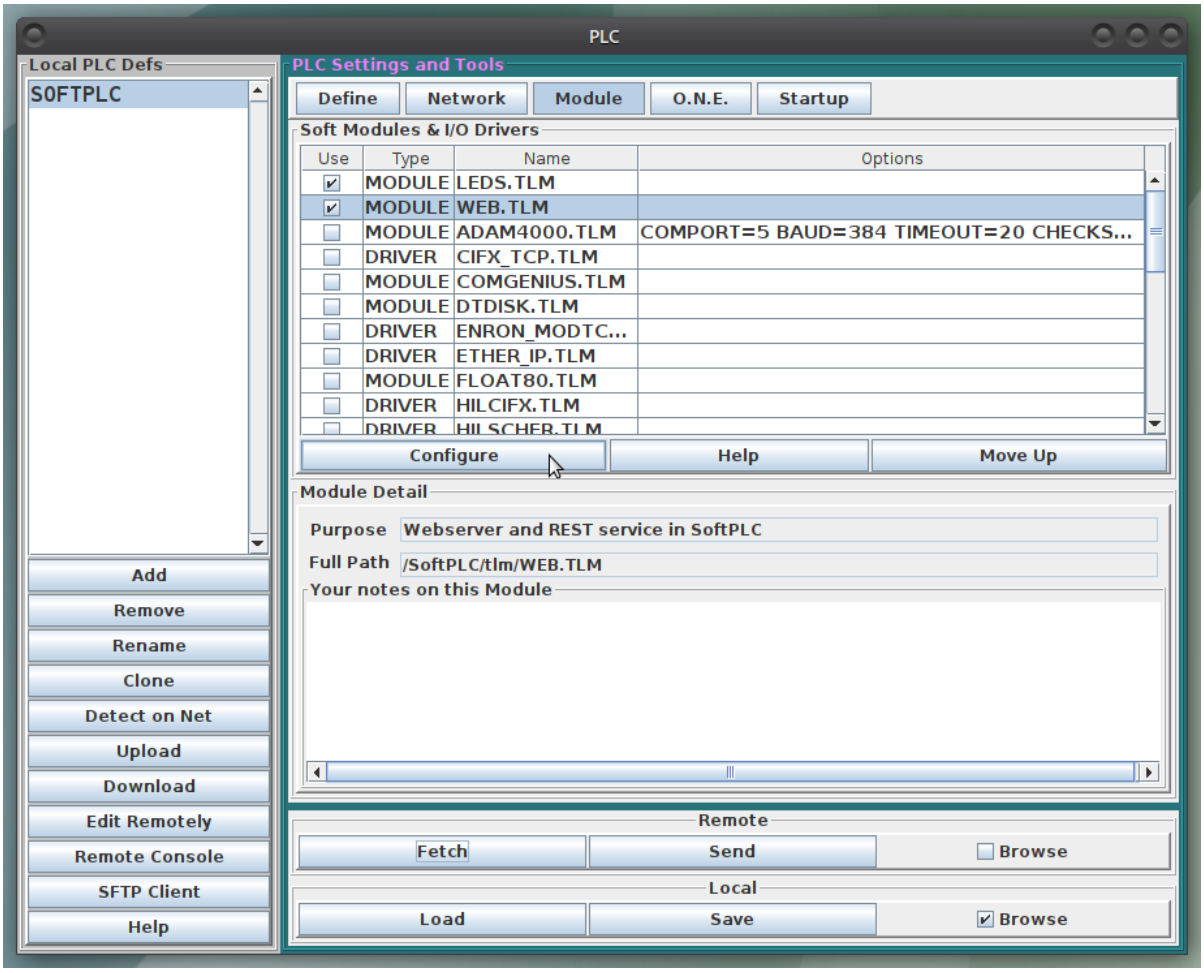
Whenever you **[Send]** a modified list of modules and/or their configuration files, you must restart (or power cycle) the SoftPLC in order for the changes to take effect.

3.1.3. Configure TLM

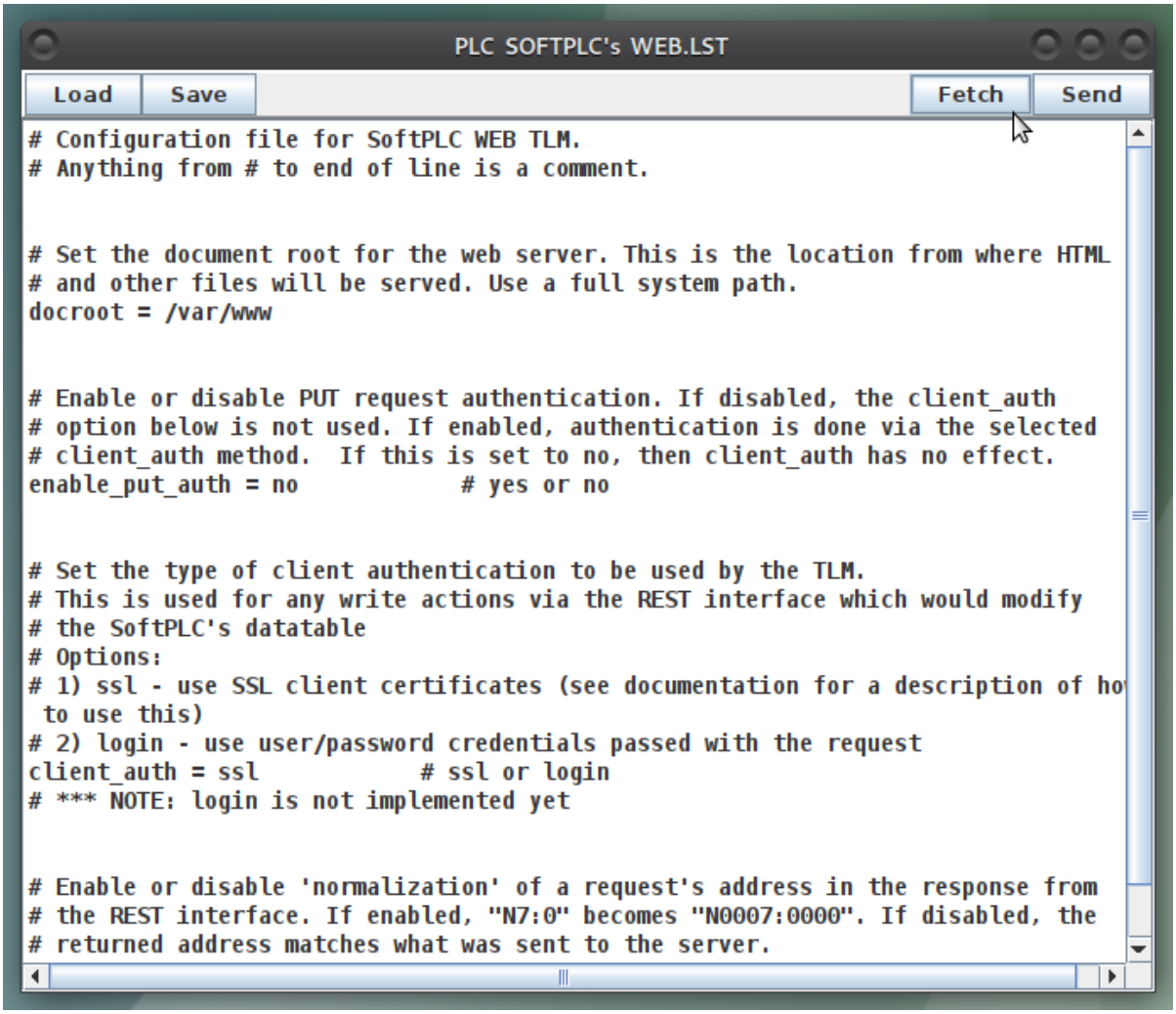
The configuration file for the WEB TLM is a text file called /SoftPLC/tlm/**WEB.LST**.

WEB.LST Configuration Editor Usage

With the WEB.TLM line highlighted/selected, click the **[Configure]** button below the list of modules to load the Configuration Editor for the WEB.LST file.



A template file is included in the SoftPLC. When connected to the SoftPLC, use the **[Fetch]** to load the template file into the Configuration Editor.



The best method for creating an WEB.LST file for your application is to start from the provided template file.

The **[Load]** button will load the WEB.LST file from the development system's disk.
The **[Save]** button will write the WEB.LST file to the development system's disk.
The **[Fetch]** button will load the WEB.LST file from the runtime system's disk.
The **[Send]** button will write the WEB.LST file to the runtime system's disk.

After you **[Send]** the configuration file, you must restart or cycle power on the SoftPLC in order for the changes to take effect.

WEB.LST Configuration File Structure

The figure below shows the structure of the WEB.LST configuration file. Depending on your application, you will need to either configure or de-activate certain sections of the template WEB.LST file.

The recommended method of de-activating sections in the template WEB.LST file is to add a "#" to the beginning of each line so that it becomes a comment.

Sample WEB.LST

```
# Configuration file for SoftPLC WEB TLM.
# Anything from # to end of line is a comment.


# Set the document root for the web server. This is the location from where HTML
# and other files will be served. Use a full system path.
docroot = /var/www


# Enable or disable PUT request authentication. If disabled, the client_auth
# option below is not used. If enabled, authentication is done via the selected
# client_auth method.  If this is set to no, then client_auth has no effect.
enable_put_auth = no            # yes or no


# Set the type of client authentication to be used by the TLM.
# This is used for any write actions via the REST interface which would modify
# the SoftPLC's datatable
# Options:
# 1) ssl - use SSL client certificates (see documentation for a description of how to use this)
# 2) login - use user/password credentials passed with the request
client_auth = ssl            # ssl or login
# *** NOTE: login is not implemented yet


# Enable or disable 'normalization' of a request's address in the response from
# the REST interface. If enabled, "N7:0" becomes "N0007:0000". If disabled, the
# returned address matches what was sent to the server.
normalize_response_addr = no    # yes or no
```

docroot

Set the document root for the web server. This is the location from where permissible files will be served. Use a full system path.

Example	Description
/var/www	All files which should be available via WEB.TLM shall be located within /var/www

enable_put_auth

Enable or disable PUT request authentication.

Possible Setting	Description
Yes (enabled)	Authentication is done via the selected **client_auth** method. See 'client_auth' below.

Possible Setting	Description
No (disabled)	The 'client_auth' option below is not used.

client_auth

Set the type of client authentication to be used by the TLM. This is used for any write actions via the REST interface which would modify the SoftPLC's datatable

Possible Setting	Description
ssl	SSL client certificates
login (not yet implemented)	User/Password credentials passed with request

normalize_response_addr

Normalization of a request's address. For example, a 'normalized' address would be "N0007:0000"; as opposed to a 'non-normalized' address of "N7:0".

Posible Setting	Description
yes	Datatable addresses in the response will be 'normalized'.
no	Datatable addresses in the response will not be 'normalized'.

Chapter 4. API Specification

This section explains the Application Programming Interface (API) supported in the REST server.

The functions defined in each section (below) are HTTP requests to the REST server in order to read/write with the SoftPLC's datatable. The expected format for information in the body of requests (and provided in the body of replies) is XML [Content-Type: text/xml]. Additional formats may be supported in the future.

'Curl' is a command line tool which generates and sends HTTP client side requests; as well as, receives responses from a server. Curl can be helpful during application development for understanding what the SoftPLC REST server expects and/or needs. Curl also helps describe how the API works, so it is used in this chapter to show the basic behaviours of each request type.

If you want to use Curl, but find it cumbersome to install on your client device, install it on the SoftPLC runtime box. Then, in a separate command line session, use it to execute the commands suggested below. After completing development of your application, it is unlikely that you will need to use CURL.

To install Curl on SoftPLC (internet access is assumed)

```
# apt update
# apt install curl
```

Once curl is installed on the runtime box, the hostname becomes **localhost**.

Quick review of the SoftPLC datatable

The SoftPLC datatable consists of a single array of sub-arrays. The top most array can hold up to 10,000 sub-arrays, called 'files'. Each file has an element type which describes all elements within that particular datatable file.

Some **element types** consist of more than one word; basically a "structure". Timers (T), counters (C), and control (R) element types all consist of 3 words each. In contrast, the most common element type is a single integer word.

A **datatable address** has the element type in the first one or two character positions. For example, PD12:3000 is a PID file (element type) whose index in the top most array is 12 and whose element number is 3000.

For elements consisting of multiple words, there **may** be a word specification attached at the end of the address following a period. For example, PD12:3000.SP is the setpoint for a PID element and refers to one "word". In the SoftPLC datatable, a **word** can be either a primitive 16 bit integer or a primitive IEEE 32 bit floating point value. In summary, PD12:3000 is an element datatable address and PD12:3000.SP is a word datatable address.

In the descriptions below, any field wrapped in < > is a place holder for a specific

use case. For example, <splc_ip> shall be replaced with the IP address specific to the SoftPLC device currently being used.

Any field wrapped in [ ] is optional.

4.1. Read Functions (HTTP GET & POST Methods)

4.1.1. GET Method

With GET, the datatable address is supplied in the URL.

Discover Available Files

```
GET http://<splc_ip>/dt
```

Fetches a listing of all available datatable files from the server. Response is sent as a web page with the listing and links available to perform a file-read on each individual file.

Read All Datatable Files

```
GET http://<splc_ip>/dt/all
```

Fetches the entire datatable.

Single File Read

The GET method can also be used to fetch data from a single datatable file, starting at an optional offset. It uses a URL that specifies a starting datatable element and a count of elements.

```
GET http://<splc_ip>/dt/<datafile>:<starting_element>[?count=<element_count>]
```

<datafile>
: is the first part of a word address like N7, PD12, etc.

<starting_element>
: is an optional starting element number. If omitted, 0 is assumed.

<element_count>
: is the number of datatable elements to fetch. If not supplied or -1, all the elements up to the end of the enclosing datatable file are returned, starting at the supplied <start_offset>.

Examples

```
from remote client:
GET http://192.100.100.60/dt/N7:3
GET http://192.100.100.60/dt/N7:3?count=3
```

```
GET http://192.100.100.60/dt/N7:3?count=-1

from local runtime:
GET http://localhost/dt/N7%3A3
```

As shown in the last example above, you may "URL encode" the ':' in between the <datafile> and <starting_element>. This decision is only pertinent to this GET method, because the ':' is in a URL. Other HTTP methods may use this character in request or response bodies without concern. The server does not require URL encoding but does support it.

CURL Examples

In CURL examples, the text may be copied to the clipboard and pasted onto the command line of the SoftPLC runtime.

GET Request

```
curl -X GET http://localhost/dt/N7:3?count=3
```

GET Response

```
<?xml version="1.0" encoding="utf-8" ?>
<files>
  <data ad="N7:3" count="3">
    0 0 0
  </data>
</files>
```

4.1.2. POST Method

With POST, the datatable addresses are supplied in the body of the HTTP request. POST operates similar to the GET method, with the added functionality of reading from more than one datatable file per request. The details of the request are specified in the HTTP request body.

```
POST http://<splc_ip>/dt
```

POST Body Format

```
<files>
  <data ad="<datafile>:<starting_element>" [count="<element_count>"]/>
  :
</files>
```

The **data** element's **ad** and **count** attributes function in the same manner as for the GET method

above. The ':' in the datatable addresses does not need to be URL encoded because it is not part of the URL. Multiple files can be read by sending additional data elements in the request.

Example Request Body

```
<files>
  <data ad="O0" count="8"/>
  <data ad="I1"/>
  <data ad="N7:33" count="10"/>
</files>
```

Example Response

Unless specifying the count attribute in the **data** XML element, the server will respond with the tail end of the datatable file starting at the given <starting_element>; if provided.

```
<files>
  <data ad="O:0000" count="8">
    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
  </data>
  <data ad="I:0001" count="7">
    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
  </data>
  <data ad="N0007:0033" count="10">
    0      0      0      0      0      0      0      0      0      0
  </data>
</files>
```

CURL Examples

In CURL examples, the request text may be copied to the clipboard and pasted onto the command line of the SoftPLC runtime.

POST Request

```
curl -X POST http://192.100.100.43/dt  -d \
        '<files> \
          <data ad="O0" count="8"/>  \
          <data ad="I1"/> \
          <data ad="N7:33" count="10"/> \
        </files>'
```

POST Response

```
<?xml version="1.0" encoding="utf-8" ?>
<files>
  <data ad="O0" count="8">
    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
  </data>
```

```
  <data ad="I1" count="7">
    0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000
  </data>
  <data ad="N7:33" count="10">
    0 0 0 0 0 0 0 0 0 0
  </data>
</files>
```

4.2. Write Functions

4.2.1. PUT Method

Writing values into the datatable is done with the HTTP PUT method. As with POST, the details of the request are placed into the body. A write can be done to a single word, multiple words in a block, and multiple words in multiple blocks by sending different **data** elements in the request body.

```
PUT http://<splc_ip>/dt
```

PUT Body Format

The HTTP request body is a single XML **files** element with one or more **data** elements within it.

```
<files>
  <data ad="<datafile>:<starting_element>" count="<element_count>">
    value ...
  </data>
  :
</files>
```

The data element's XML attributes specify the file type and number <datafile>, element offset at which to begin writing <starting_element>, and number of elements to be written <element_count>. The actual values to be written are specified in plain text inside the data element, separated by whitespace. For a PUT request to be valid, all of this information must be provided (and match correctly, so the number of values must match the count attribute). The example below shows a PUT to multiple elements in the same file, one of which is at a non-continuous location from the others.

```
<files>
  <data ad="N7:0" count="2">42 0xf73d</data>
  <data ad="N7:7" count="1">3</data>
</files>
```

Response Format

The response to a PUT request is identical to what would be received as a response by doing a POST for the values that were in the request. Thus, the sample below is the response that would be received for the PUT example above.

```
<files>
  <data ad="N0007:0000" count="2">
    42     63293
  </data>
  <data ad="N0007:0007" count="1">
    3
  </data>
</files>
```

Request body accepts hex values but converts them to decimal in the response.

CURL Examples

In CURL examples, the request text may be copied to the clipboard and pasted onto the command line of the SoftPLC runtime.

PUT Request

```
curl -X PUT http://192.100.100.43/dt    -d \
    '<files> \
       <data ad="n7;0" count="2">0x00FF 2</data> \
       <data ad="N7:7" count="1">3</data> \
    </files>'
```

PUT Response

```
<?xml version="1.0" encoding="utf-8" ?>
<files>
  <data ad="n7;0" count="2">
    255 2
  </data>
  <data ad="N7:7" count="1">3</data>
</files>
```

PUT Body Format (Word level)

```
<files>
  <data ad="<datafile>:<starting_element>.<word>">
    value
  </data>
</files>
```

The XML attributes are specified similarly to those described in section PUT Body Format. The datatable address must be a word address (notice the .<word>). The word portion of the "ad" attribute specifies which word in the element is to be written. The actual value to be written is specified in plain text inside the XML <data> element. Note that each <data> element can only specify access to a single word value, therefore there is no "count" attribute needed for this format. If a count is provided it will be ignored.

Example

```
<files>
  <data ad="T4:0.PRE">4000</data>
</files>
```

4.3. Tag Substitution

In place of using addresses specified as <datafile>:<starting_element> in the requests, tags may be used instead. All used tags must be defined in the descriptor table and map to their respective addresses. This applies to all request types.

Example GET URL

```
GET http://192.100.100.60/dt/timer1?count=1
```

Example POST Body

```
<files>
  <data ad="timer1" count="1"/>
</files>
```

Example PUT Body

```
<files>
  <data ad="important_int" count="1">555</data>
</files>
```

The response to a request with tags will be similar to a request with addresses, except addresses will be switched out with tags.