

---

# **rawdatx Documentation**

***Release 0.1***

**Chris Petrich**

**Dec 26, 2018**



---

## Contents

---

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Usage</b>	<b>5</b>
<b>3</b>	<b>Background</b>	<b>7</b>
3.1	Availability . . . . .	7
3.2	Author . . . . .	7
<b>4</b>	<b>Contents</b>	<b>9</b>
4.1	XML Definition File . . . . .	9
4.2	Document Type Definition . . . . .	12
4.3	Functions Exported . . . . .	13
4.4	Processing Configuration . . . . .	13
<b>5</b>	<b>Indices and tables</b>	<b>15</b>



rawdatx is a Python 2.7, 3.4, 3.5 converter that generates Excel xlsx files from TOA5 comma-separated text files produced by Campbell Scientific LoggerNet. Sensor input, processing instructions, and output structure are specified in a single XML Definition File that also serves as documentation.



# CHAPTER 1

---

## Installation

---

The following prerequisites need to be installed:

- Python 2.7, 3.4, or 3.5
- numpy 1.9 or higher
- xlswriter

optionally (recommended):

- lxml
- asteval

The easiest way to install rawdatx is through pip: `pip install rawdatx`

Alternatively, download the latest version from the repository <https://github.com/cpetrich/rawdatx> and install with `python setup.py install`.





## CHAPTER 2

---

### Usage

---

To convert a TOA5 file to XLSX, run the following script:

```
import rawdatx.read_TOA5 as read_raw_data
import rawdatx.process_XML as process_XML

config = './config.cfg'
read_raw_data.main(config)
process_XML.main(config)
```

Input and output files are specified in an UTF-8 encoded configuration file `config.cfg`:

```
[RawData]
raw_data_path      = ./raw-data/
mask               = CR1000_*.dat
logger_time_zone   = UTC+1

[Metadata]
Project            = My project name

[Files]
xml_map_path       = ./
xml_map            = data_map.xml
data_path          = ./
processed_data_xlsx = processed_data.xlsx
xml_dtd_out        = data_map.dtd
raw_data           = consolidated_raw_data.npy
processed_data_npy  = processed_data.npy
```

The `[RawData]` section specifies the location of the logger input files, the `[Metadata]` section defines metadata entries copied into the XLSX file, and the `[Files]` section specifies path and file names of output and intermediate files (`data_path`) and input XML Definition File (`xml_map_path` and `xml_map`).

The XML Definition File (`data_map.xml`) may look like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
<measurements from="2015/05/03 11:45">
  <group name="Logger">
    <map name="Battery Voltage" unit="V" src="Batt_V" />
    <map name="Internal Temperature" unit="°C" src="T_panel" />
  </group>
  <group name="Weather">
    <map name="Air Temperature" unit="°C" src="T_air" />
    <map name="Relative Humidity" unit="%" src="RH" />
    <map name="Wind Speed" unit="m/s" src="Wind_speed" />
    <map name="Wind Direction" unit="°" src="Wind_direction" />
  </group>
</measurements>
```

See also examples and test files in the repository at <https://github.com/cpetrich/rawdatx>.

### 3.1 Availability

The code is available under the MIT license. The project is hosted at <https://github.com/cpetrich/rawdatx> and packages are available on PyPI at <https://pypi.python.org/pypi/rawdatx/>. Documentation is available at <https://rawdatx.readthedocs.org/>.

### 3.2 Author

Chris Petrich



### 4.1 XML Definition File

The formal Document Type Definition of the XML is given in the section *below*.

#### 4.1.1 Minimal example

An XML file for a simple deployment of a weather station is shown below.

- To produce output, an XML Definition File must contain the root `measurements` element and at least one `map` element inside a parent `group` element.
- Each `group` element must have a `name` attribute. The `name` attribute may be an empty string (i.e., `name=""`).
- In the example below, a global `from` attribute is specified (optional) to limit data output to the time after all sensors were in place and connected to the logger.

```
<?xml version="1.0" encoding="UTF-8" ?>
<measurements from="2015/05/03 11:45">
  <group name="Logger">
    <map name="Battery Voltage" unit="V" src="Batt_V" />
    <map name="Internal Temperature" unit="°C" src="T_panel" />
  </group>
  <group name="Weather">
    <map name="Air Temperature" unit="°C" src="T_air" />
    <map name="Relative Humidity" unit="%" src="RH" />
    <map name="Wind Speed" unit="m/s" src="Wind_speed" />
    <map name="Wind Direction" unit="°" src="Wind_direction" />
  </group>
</measurements>
```

### 4.1.2 Example containing all elements and attributes

Here is a hypothetical example using all defined elements and attributes in some form.

- Note in particular that *each* element may contain time validity attributes from, until, except-from and except-until.
- If **any** element uses an until or except-until attribute then attribute until-limit must be set to one of inclusive or exclusive in the measurement element. The default value is disallowed to avoid ambiguity in intent.
- The sole purpose of set elements is to propagate time validity attributes to their children.
- The actual valid time of an output variable is the period that is not excluded by time validity attributes of itself and all parent elements combined.
- Time limitation is applied to the **result** of a map or def element rather than the **input** specified in the src attribute. See **case below** for an example where this is significant.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- "measurements" is the root element -->
<measurements name="Campaign" from="2014/09/01 12:00" until="2015/03/07 17:00" until-
→limit="inclusive" /> <!-- "measurements" is mandatory -->
  <!-- we may choose to enclose several "group" elements
    in a "set" if they share time validity attributes -->
  <set from="2014/10/01" comment="'comment' values are ignored"> <!-- "set" is_
→optional -->
    <!-- data are placed together in
      "group"s that share a common title -->
    <group name="Weather"> <!-- "group" is mandatory, "name" attribute must be_
→specified -->
      <!-- within a "group" several data fields
        may be enclosed as a "set" if they
        share time validity attributes -->
      <set except-from="2015/01/30 04:00" expept-until="2015/02/01 08:00"> <!--
→"set" is optional -->
        <!-- actual variable output is defined by
          "map" elements. -->
        <map name="Air Temperature" unit="K" is="T_C+273.15" />
        <!-- functions or constants are defined
          in "def" elements if they do not
          produce output. "Name" and "unit" attributes
          are ignored.
          The special variable "SRC" (all upper case)
          refers to the source variable specified in
          the "src" attribute of the current "map" or
          "def" element. -->
        <def var="T_C" is="(SRC-32)*5/9." src="T_air_in_F" />
        <!-- note that the order of variable definition
          is irrelevant, i.e. the preceeding "map" element
          refers to variable "T_C" defined later. -->
      </set>
    </group>
  </set>
</measurements>
```

### 4.1.3 Example with function definition

- Functions are defined in `<def />` elements. The function signature and function expression are placed in the value of the `var` and `is` attribute, respectively.
- The expression in the `is` attribute value must be a valid Python expression.
- Due to the use of the `asteval` library, lambda expressions are not allowed.
- Global variables take precedent over local variables (may change in future versions). Recommendation: do not use function parameters that coincide with names defined through `var` attributes.

```
<?xml version="1.0" encoding="UTF-8" ?>
<measurements>
  <group name="Weather">
    <map name="Air Temperature" unit="°C" var="T_C" src="T_air_C" />
    <map name="Air Temperature" unit="°F" is="C_to_F(T_C)" />
    <map name="Soil Temperature" unit="°C" var="T_soil" src="T_soil_C" />
    <map name="Water Temperature" unit="°C" var="T_water" src="T_water_C" />
    <def var="difference(T1, T2)" is="abs(T1-T2)" /> <!-- a rather unusual place_
↳for this definition -->
  </group>
  <group name="Processed Weather">
    <map name="Relative Air Temperature" unit="°C" is="relative_T(T_air)" />
    <map name="Relative Soil Temperature" unit="°C" is="relative_T(T_soil)" />
    <map name="Absolute Soil-Air Temperature Difference" unit="°C" is=
↳"difference(T_C, T_soil)" />
  </group>
  <group name="function definitions">
    <def var="C_to_F(T_degC)" is="T_degC*9/5+32" />
    <def var="relative_T(T_base)" is="T_base-T_water" />
  </group>
</measurements>
```

In this example, the global variable names `T_C`, `T_water`, and `T_soil` should not be used as function parameters (`T_degC`, `T_base`, `T1`, `T2`). However, they can be used as global variables in the function body.

---

**Note:** Function definitions may be placed throughout the document in any order.

---

### 4.1.4 Example of *in-situ* calibration

In this example, a zero-point calibration is performed on a sensor based on the average reading shortly after deployment.

This is an example where it matters that time limitation is applied to the **result** of a calculation: a `def` element

```
<def val="p_1_offset" is="mean(SRC)" src="p_1" from="2015/01/01" until="2015/01/02" />
```

does **not work as intended** because `mean()` is calculated over the entire time series of `p_1` (unless constraint by parents) while the **output** is time limited to the period from `from` until `until`. Instead, we have to split the offset definition into two expressions:

```
<def val="p_1_masked" src="p_1" from="2015/01/01" until="2015/01/02" />
<def val="p_1_offset" is="mean(p_1_masked)" />
```

Complete example:

```

<?xml version="1.0" encoding="UTF-8" ?>
<measurements from="2015/05/03 11:45" until-limit="inclusive">
  <group name="Logger">
    <map name="Battery Voltage" unit="V" src="Batt_V" />
    <map name="Internal Temperature" unit="°C" src="T_panel" />
  </group>
  <group name="Load">
    <map name="Average" unit="kPa" is="0.5*(P1+P2)" />
    <map name="Sensor 1" unit="kPa" var="P1" is="SRC-p_1_offset" src="p_1" />
    <map name="Sensor 2" unit="kPa" var="P2" is="SRC-mean(p_2_masked)" src="p_2" /
  </group>
  <group name="zero-point definitions" >
    <!-- "group" does not produce output because it
         contains no "map" elements -->
    <set from="2015/05/03 11:45" until="2015/05/03 12:45">
      <def val="p_1_masked" src="p_1" />
      <def val="p_2_masked" src="p_2" />
    </set>
    <def val="p_1_offset" is="mean(p_1_masked)" />
  </group>
</measurements>

```

## 4.2 Document Type Definition

The formal Document Type Definition (DTD) of the XML Definition File is:

```

<!ELEMENT measurements ((set|group)*)>
<!ELEMENT set ((set|group|map|def)*)>
<!ELEMENT group ((set|map|def)*)>
<!ELEMENT map EMPTY>
<!ELEMENT def EMPTY>
<!ENTITY % may-have-name "name CDATA #IMPLIED">
<!ENTITY % must-have-name "name CDATA #REQUIRED">
<!ENTITY % until-mode "until-limit CDATA #IMPLIED">
<!ENTITY % inheritable "from CDATA #IMPLIED until CDATA #IMPLIED except-from CDATA
↳#IMPLIED except-until CDATA #IMPLIED">
<!ENTITY % definition "var CDATA #IMPLIED is CDATA #IMPLIED src CDATA #IMPLIED unit_
↳CDATA #IMPLIED">
<!ENTITY % common "%inheritable; comment CDATA #IMPLIED">
<!ATTLIST measurements %may-have-name; %until-mode; %common;>
<!ATTLIST group %must-have-name; %common;>
<!ATTLIST set %may-have-name; %common;>
<!ATTLIST map %must-have-name; %definition; %common;>
<!ATTLIST def %may-have-name; %definition; %common;>

```

Note that the DTD is more permissive than the XML interpreter:

- each map element has to be decendent of a group element, either directly or indirectly.
- an until-limit attribute is required in the measurements element if **any** element in the document uses an until or except-until attribute.



## 4.3 Functions Exported

The following functions and constants are available in the evaluation environment of the values of the `is` attribute.

Functions from `numpy`:

- `ln()`, `log10()`, `exp()`
- `fabs()`, `abs()`: `abs()` is an alias for `numpy.fabs()`
- `sign()`
- `sin()`, `cos()`, `tan()`, `arctan()`, `arctan2()`
- `mean()`, `sum()`, `min()`, `max()` (each function mapping to the corresponding function `numpy.nanmin()` etc)
- `round()`, `isnan()`
- `where()`, `len()`: functions emulating `numpy` behavior

Convenience functions:

- `merge(vector1, vector2)`: returns `numpy.where(vector1==vector1, vector1, vector2)`
- `replace_value_with_NaN(vector, value)`: returns `vector[vector==value]=NaN`
- `replace_time_with_NaN(vector, list_of_time_strings)`: returns `vector` with values recorded at the specified times replaced by `NaN`. Example: `is="replace_time_with_NaN(T1, ['2015/03/01 11:00', '2015/03/01 11:05'])`
- `in_date_range(start,end)`: returns `vector` with `True` for all times between `start` (inclusive) and `end` (exclusive).

Also defined:

- `None`: evaluates to Python value `None`
- `PI`: evaluates to  $\pi$
- `NaN`: evaluates to `float('nan')`
- `float()`: evaluates Python function `float()`

Experimental:

- `remove_spikes(vector)`: heuristic function used to remove outliers. Implementation of this function is subject to change.

## 4.4 Processing Configuration

Basic configuration parameters, in particular directory paths, are stored in a configuration file used by both data extraction script and the XML interpreter.

The file is UTF-8 encoded and follows the MS INI format:

```
[RawData]
raw_data_path      = ../raw-data/
mask               = CR1000_*.dat
logger_time_zone   = UTC+1

[Metadata]
Project            = My project name
```

(continues on next page)

(continued from previous page)

```
Web Page      = http://my-project.org/
File Content  = Data acquired during My Project
Owner        = Myself
Contact      = me@my-email.com
Comment      = Data are provided without warranty of fitness for a particular purpose.

[Files]
data_path     = ./
xml_dtd_out   = data_map.dtd
raw_data      = consolidated_raw_data.npy
processed_data_xlsx = processed_data.xlsx
processed_data_npy = processed_data.npy
xml_map_path  = ./
xml_map       = data_map.xml
```

### 4.4.1 RawData

The RawData section is used only by the data extraction tool.

- `raw_data_path` specifies the path to the Campbell Scientific TOA5 raw data files generated by LoggerNet.
- `mask` is the file name mask (glob) of the TOA5 raw data files that should be imported.
- `logger_time_zone` is a string that is copied into the output file

### 4.4.2 Metadata

The Metadata section is used by the XML interpreter. The keys and values are copied into the output XLSX file ahead of the data table.

### 4.4.3 Files

The Files section is used by both raw data extraction tool and XML interpreter:

- `data_path` specifies where intermediate and final files will be stored.
- `raw_data` specifies the name of the file generated by the TOA5 raw data extraction tool.
- `processed_data_xlsx` and `processed_data_npy` are the names of the files that store the result of XML interpreter.
- `xml_dtd_out` names the file the XML interpreter stores the DTD in.
- `xml_map_path` (optional) is the path to the XML Data File. If not specified, `data_path` is assumed.
- `xml_map` is the input XML Data File for the XML interpreter.

## CHAPTER 5

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`