Utils

Was ist die utils.py?

Die utils.py umfasst zwei wichtige Funktionen. Einmal die Klasse DeviceTree und die Funktion xml_to_dict.

Die Klasse DeviceTree umfasst mehrere Methoden zum erstellen einer XML Baumstruktur und dem Hinzufügen eines Timestamps.

Diese Methoden sind “Arbeitsmethoden” und führen die gesammelten Daten der Wrapper in eine Baumstruktur um, damit diese weiterverarbeitet werden kann.

Die Funktion xml_to_dict wandelt eine gegebene XML Baumstruktur in ein Python Dictionary um.

Code

```
# -*- coding: utf-8 -*-

import logging
from sys import stderr
from time import mktime, strptime, gmtime

from lxml import etree

import config

logger = logging.getLogger(__name__)

class DeviceTree:
   def __init__(self, dev, subdev):
       """ __init__
        Konstruktur. Initialisiert eine XML Baumstruktur.
        @param dev Device.
        @param subdev Subdevice.
        """
       logger.debug('CALL: __init__(%s, %s)', dev, subdev)
       self._dev = etree.Element('device')
       self._dev.attrib['name'] = unicode(dev)
       logger.debug('_dev.attrib->name = %s', self._dev.attrib['name'])
       self._subdev = etree.SubElement(self._dev, 'subdev')
       self._subdev.text = unicode(subdev)
       logger.debug('_subdev.text = %s', self._subdev.text)

   def add_return_value(self, key, value):
       """ add_return_value
        Fuegt der XML Struktur einen Kindknoten hinzu.
        @param key Tag.
        @param value Text.
        """
       logger.debug('CALL: add_return_value(%s, %s)', key, value)
       ret = etree.SubElement(self._dev, unicode(key))
       ret.text = unicode(value)
       logger.debug('ret.text = %s', ret.text)

   def add_subelement(self, tag, **children):
```

```
    """ add_subelement
     Fuegt der XML Struktur einen Teilbaum hinzu.
     @param tag Tag.
     @children Kindknoten.
     """
    logger.debug('CALL: add_subelement(%s, %s)', tag,
            children.items())
    subelement = etree.SubElement(self._dev, unicode(tag))
    for key, value in children.items():
        etree.SubElement(subelement, unicode(key)).text = \
                unicode(value)

def add_timestamp(self, begin=None, end=None):
    """ add_timestamp
     Fuegt der XML Struktur einen Timestamp hinzu.
     @param begin Zeitpunkt oder None.
     @param end Zeitpunkt oder None.
     """
    logger.debug('CALL: add_timestamp(%s, %s)', begin, end)
    timestamp = etree.SubElement(self._dev, 'timestamp')
    if begin == None and end == None:
        timestamp.text = unicode(DeviceTree.make_timestamp())
        logger.debug('timestamp.text = %s', timestamp.text)
    elif begin != None and end == None:
        timestamp.text = unicode(begin)
        logger.debug('timestamp.text = %s', timestamp.text)
    elif begin != None and end != None:
        timestamp_begin = etree.SubElement(timestamp, 'begin')
        timestamp_begin.text = unicode(begin)
        logger.debug('timestamp_begin.text = %s', timestamp_begin.text)
        timestamp_end = etree.SubElement(timestamp, 'end')
        timestamp_end.text = unicode(end)
        logger.debug('timestamp_end.text = %s', timestamp_end.text)
    else:
        logger.warning('Timestamp(s) nicht gesetzt')
        return
    logger.info('Timestamp(s) wurden gesetzt')

def get_tree(self):
    """ get_tree
     Gibt die XML Baumstruktur zurueck.
     @return XML Baumstruktur.
     """
    logger.debug('CALL: get_tree()')
    return self._dev

@staticmethod
def make_timestamp(time=None, format_=None):
    """ make_timestamp
     Erstellt einen UNIX Timestamp.
     @param time String, Datetime oder None.
     @param format_ String oder None.
     @return UNIX Timestamp.
     """
    logger.debug('CALL: make_timestamp(%s, %s)', time, format_)
    if time == None and format_ == None:
        struct = gmtime()
    else:
        try:
            # beide Argumente muessen vom Typ str sein
            struct = strptime(time, format_)


        except Exception as e:
            # ... war wohl nicht so; koennte datetime sein
            print >> stderr, e
            logger.warning('ungueltige Strings: %s', e)
```

```
                try:
                    timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
                except Exception as e:
                    # ... doch nicht; verwende aktuellen Timestamp
                    print >> stderr, e
                    logger.warning('Unbekannter timestamp: %s', e)
                    struct = gmtime()
                else:
                    struct = strptime(timestamp, '%Y-%m-%d %H:%M:%S')
        logger.debug('struct = %s', struct)
        logger.info('Timestamp wurde erstellt')
        return mktime(struct)

def xml_to_dict(xml):
    """ xml_to_dict
     Erstellt aus einer vorhandenen XML Struktur ein Python dictionary.
     @param xml XML Struktur.
     @return Python dictionary.
     """
    ret = {}
    try:
        ret['device'] = xml.get('name')
        for elem in list(xml):
            if len(list(elem)) > 0:
                tmp = ret.get(unicode(elem.tag))
                if tmp == None:
                    # Eintrag noch nicht vorhanden -> anlegen
                    items = {}
                    for item in list(elem):
                        items[unicode(item.tag)] = unicode(item.text)
                    ret[elem.tag] = [items]
                else:
                    # Eintrag schon vorhanden -> erweitern
                    items = {}
                    for item in list(elem):
                        items[unicode(item.tag)] = unicode(item.text)
                    tmp.append(items)
            else:
                ret[elem.tag] = elem.text
    except Exception as e:
        print >> stderr, e
        logger.error('Umformung konnte nicht durchgefuehrt werden: %s', e)
        ret = None
    finally:
        return ret

# vim: set sts=4 sw=4 et:
```