SPRINT-2

Data Publish to IOTDevice

SPRINT-2	14 November 2022
Project name	IoT Based Safety Gadget for Child Safety Monitoring and Notification
Team Id	PNT2022TMID24203

```
from_future
import
absolute_import,
division,
unicode_literals


        try:
        from collections.abc import
        MutableMapping except
        ImportError: # Python 2.7 from
        collections import
        MutableMapping from
        xml.dom import minidom,
        Node import weakref

        from . import base from ..
        import constants from
        ..constants import
        namespaces from .._utils
        import moduleFactoryFactory
```

```
  def
  getDomBuilder(DomImplemen
  tation): Dom =
  DomImplementation

  class
  AttrList(MutableMappi
  ng):    def__init__(self,
  element):  self.element
  = element

                  def_iter_(self): return
iter(self.element.attributes.keys())

  def_setitem_(self, name, value): if
  isinstance(name, tuple): raise
  NotImplementedError else: attr =
  self.element.ownerDocument.creat
  eAttribute(name) attr.value = value
  self.element.attributes[name] = attr

  def_len_(self): return
  len(self.element.attribute
  s)

  def items(self):
  return
  list(self.element.attributes.items())
```

```
def values(self): return
list(self.element.attributes.values
())

def__getitem__(self, name): if
isinstance(name, tuple): raise
NotImplementedError else:
return
self.element.attributes[name].v
alue

def_delitem_(self, name):
if isinstance(name,
tuple): raise
NotImplementedError
else: del
self.element.attributes[n
ame]

class
NodeBuilder(base.Node)
: def_init_(self, element):
base.Node._init_(self,
element.nodeName) self.element =
element

namespace = property(lambda
self: hasattr(self.element,
"namespaceURI") and
```

```
        self.element.namespaceURI  or
        None)

    def appendChild(self, node):
        node.parent = self
        self.element.appendChild(node.elem
        ent)

        def insertText(self, data,
        insertBefore=None): text =
        self.element.ownerDocument.create
        TextNode(data) if insertBefore:
        self.element.insertBefo
        re(text,
        insertBefore.element)
        else:
        self.element.appendCh
        ild(text)

        def insertBefore(self, node,
        refNode):
        self.element.insertBefore(no
        de.element,
        refNode.element)
        node.parent = self

        def removeChild(self, node): if
        node.element.parentNode ==
        self.element:
```

```
self.element.removeChild(node.
element) node.parent = None

def reparentChildren(self,
newParent): while
self.element.hasChildNodes():
child = self.element.firstChild
self.element.removeChild(chil
d)
newParent.element.appendCh
ild(child) self.childNodes = []

def getAttributes(self):
return
AttrList(self.element)

def setAttributes(self,
attributes): if attributes: for
name, value in
list(attributes.items()): if
isinstance(name, tuple): if
name[0] is not None:
qualifiedName = (name[0] + ":" +
name[1]) else:
qualifiedName = name[1]
self.element.setAttributeNS(name[2],
qualifiedName, value) else:
```

```
        self.element.setAttribute( name,
        value) attributes =
        property(getAttributes, setAttributes)

        def cloneNode(self):
        return
        NodeBuilder(self.element.cloneNode
        (False))

        def hasContent(self):
    return self.element.hasChildNodes()

        def
        getNameTuple(self)
        : if self.namespace
        is None:
        return
        namespaces["html"],
        self.name else: return
        self.namespace, self.name
        nameTuple =
        property(getNameTuple)

        class
        TreeBuilder(base.TreeBuilde
        r): # pylint:disable=unused-
        variable def
        documentClass(self):
        self.dom =
```

```
Dom.getDOMImplementation().cre
ateDocument(None, None, None)
return weakref.proxy(self)

def insertDoctype(self,
token): name =
token["name"] publicId
= token["publicId"]
systemId =
token["systemId"]

domimpl =
Dom.getDOMImplementation()
doctype =
domimpl.createDocumentType(na
me, publicId, systemId)
self.document.appendChild(NodeB
uilder(doctype)) if Dom ==
minidom: doctype.ownerDocument
= self.dom

def elementClass(self, name,
namespace=None): if namespace is
None and self.defaultNamespace is
None:
node =

self.dom.createElement(name) else:

node =
```

```
        self.dom.createElementNS(namesp
        ace, name) return
        NodeBuilder(node)

        def commentClass(self, data):
        return
        NodeBuilder(self.dom.createComm
        ent(data))

    def fragmentClass(self):
        return
        NodeBuilder(self.dom.createDocume
        ntFragment())

        def appendChild(self, node):
        self.dom.appendChild(node.element
        )

        def testSerializer(self,
        element): return
        testSerializer(element)

        def getDocument(self):
        return self.dom

        def getFragment(self): return
        base.TreeBuilder.getFragment(self).e
        lement

        def insertText(self, data,
        parent=None):
```

```
    data = data if
    parent !=
    self:
    base.TreeBuilder.insertText(self,
    data, parent) else:
    # HACK: allow text nodes as children
    of the document node if
    hasattr(self.dom,
    '_child_node_types'): #
    pylint:disable=protected-access if
    Node.TEXT_NODE not in
    self.dom._child_node_types:
    self.dom._child_node_types =
    list(self.dom._child_node_types)
    self.dom._child_node_types.append(N
    ode.TEXT_NODE)
    self.dom.appendChild(self.dom.create
    TextNode(data))

    implementation =
    DomImplementation name =
    None

def testSerializer(element):
    element.normali
    ze() rv = []
```

```
def serializeElement(element,
indent=0): if element.nodeType ==
Node.DOCUMENT_TYPE_NODE: if
element.name: if element.publicId
or element.systemId:
publicId = element.publicId or ""
systemId = element.systemId or ""
rv.append("""|%s<!DOCTYPE %s
"%s" "%s">""" % (' ' * indent,
element.name, publicId, systemId))
else:
rv.append("|%s<!DOCTYPE %s>"
% (' ' * indent, element.name))
else:
rv.append("|%s<!DOCTYPE >" % (' ' *
indent,)) elif element.nodeType ==
Node.DOCUMENT_NODE:
rv.append("#document") elif
element.nodeType ==
Node.DOCUMENT_FRAGMENT_
NODE: rv.append("#document-
fragment") elif
element.nodeType ==
Node.COMMENT_NODE:
rv.append("|%s<!-- %s -->" % (' '
* indent, element.nodeValue))
elif element.nodeType ==
```

```
Node.TEXT_NODE:
rv.append("|%s\"%s\"" % (' ' *
indent, element.nodeValue))
else:
if (hasattr(element,
"namespaceURI") and
element.namespaceURI is not None):
name = "%s %s" %
(constants.prefixes[element.na
mespaceURI],
element.nodeName) else:
name = element.nodeName
rv.append("|%s<%s>" % (' ' *
indent, name)) if
element.hasAttributes():
attributes = [] for i in
range(len(element.attributes)
):
attr =
element.attributes.item(
i) name =
attr.nodeName value =
attr.value ns =
attr.namespaceURI if ns:
name = "%s %s" %
(constants.prefixes[ns],
attr.localName) else:
```

```
        name = attr.nodeName
        attributes.append((name, value))

        for name, value in sorted(attributes):
        rv.append('|%s%s="%s"' % (' ' *
        (indent + 2), name,


        value)) indent += 2 for

        child in

        element.childNodes:

        serializeElement(child,

        indent)

        serializeElement(eleme

        nt, 0) return

        "\n".join(rv) return

        locals()


# The actual means to get a module!
                                getDomModule =
    moduleFactoryFactory(getDomBuilder)
```