

**Project Development Phase**  
**SPRINT-2**

Date	29 October 2022
Team ID	PNT2022TMID17405
Project Name	Project - IOT Based Safety Gadget for Child Safety Monitoring&Notification

```
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(DomImplementation):
```

```
    Dom = DomImplementation
```

```
class AttrList(MutableMapping):

    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.createAttribute(name)
            attr.value = value
            self.element.attributes[name] = attr

    def __len__(self):
        return len(self.element.attributes)

    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].value

    def __delitem__(self, name):
if isinstance(name, tuple):
raise NotImplementedError

    else:

        del self.element.attributes[name]

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.element)

    def insertText(self, data, insertBefore=None):

        text = self.element.ownerDocument.createTextNode(data)

        if insertBefore:

            self.element.insertBefore(text, insertBefore.element)

        else:

            self.element.appendChild(text)

```

```

def insertBefore(self, node, refNode):

    self.element.insertBefore(node.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(child)
newParent.element.appendChild(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.TreeBuilder): # pylint:disable=unused-variable
def documentClass(self):
    self.dom = Dom.getDOMImplementation().createDocument(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(name, publicId, systemId)
self.document.appendChild(NodeBuilder(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(namespace, name)

    return NodeBuilder(node)

    def commentClass(self, data):

        return NodeBuilder(self.dom.createComment(data))

    def fragmentClass(self):

        return NodeBuilder(self.dom.createDocumentFragment())

    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).element

    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(Node.TEXT_NODE)
self.dom.appendChild(self.dom.createTextNode(data))

implementation = DomImplementation
name = None

```

```

def testSerializer(element):
    element.normalize()

    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.namespaceURI],
                           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(element, 0)

    return "\n".join(rv)

return locals()

```

# The actual means to get a module!

getDomModule = moduleFactoryFactory(getDomBuilder)