Kommentarübersicht zu <RAILING.pdf>

Erstellt am 19/1/2021 um 21:50:10

```python
import OpenLxApp as lx
import OpenLxCmd as cmd
import OpenLxUI as ui
import Base
import Geom
import Topo

lxStr = Base.StringTool.toString

GUID_CLASS = Base.GlobalId("{20210119-DEAD-C0DE-C1A5-000000000001}")
GUID_SCRPT = Base.GlobalId("{20210119-DEAD-C0DE-5C17-000000000001}")


class EditMode:
    def __init__(self, doc):
        if doc is None:
            raise RuntimeError("Document is None.")

        self._doc = doc
        self._exitEditing = False

    def __enter__(self):
        if not self._doc.isEditing():
            self._doc.beginEditing()
            self._exitEditing = True
        else:
            self._exitEditing = False

    def __exit__(self, exc_type, exc_val, exc_tb):
        if self._exitEditing:
            self._doc.endEditing()
            ui.UIApplication.getInstance().getUIDocument(self._doc).getSelection().forceUpdate()
            self._doc.recompute()


class RailingAxis(lx.Railing):
    def __init__(self, aArg):
        lx.Railing.__init__(self, aArg)
        self.registerPythonClass("RailingAxis", "OpenLxApp.Railing")

        self.setBoundingBoxEnabled(False)
                                      [1]
        # Header and Group
        self.setPropertyHeader(lxStr("Railing with Axis"), -1)                                                  # TODO: Replace -1 with the corresponding translatorId
        self.setPropertyGroupName(lxStr("Railing with Axis"), -1)                                               # TODO: Replace -1 with the corresponding translatorId

        # Property "_subdivisions"
        self._subdivisions = self.registerPropertyInteger("_subdivisions", 20, lx.Property.VISIBLE, lx.Property.EDITABLE, -1)   # TODO: Replace -1 with the corresponding translatorId
                                                        [2]                                                 [3]
        # Property "_representation"
        self._representation = self.registerPropertyEnum("_representation", 1, lx.Property.VISIBLE, lx.Property.EDITABLE, -1)   # TODO: Replace -1 with the corresponding translatorId
        self._representation.setEmpty()                          [4]
        self._representation.addEntry(lxStr("Axis"))        # Index 0
        self._representation.addEntry(lxStr("SolidModel"))  # Index 1

    def _setAxisCurve(self, axisCurve):
        with EditMode(self.getDocument()):
            """
            Here we set the Axis
            """
            ok = self.setAxisRepresentation(axisCurve)
                                                   [5]
            """
            Recreate the "MultiGeo" based on the Axis
            """
            self._updateSolidModel()
            return ok

    def _switchRepresentations(self, index):
        with EditMode(self.getDocument()):
            if index == 0:                          # Index 0
                self.showAxisRepresentation()
            else:        [6]
                self.showSolidModelRepresentation()

                """
                Recreate the "MultiGeo" based on the Axis
                """
                self._updateSolidModel()
                                      [7]
    def _updateSolidModel(self):
        """
        Update SolidModel only when it is really shown
        """
        if self._representation.getValue() == 0:            # Index 0
            return

        with EditMode(self.getDocument()):
            self.removeSubElements()

            """
            Here we get the Axis
            """
            axisCurve = self.getAxisRepresentation()
                                                   [8]
            """
            Calculate the position of each step on the curve and create SubElements
            """
            edge = None
            if axisCurve:
                edge = Topo.EdgeTool.join(Topo.WireTool.getEdges(Topo.ShapeTool.isSingleWire(axisCurve.computeShape(False))))

            if edge:
                length = Topo.EdgeTool.getLength(edge)
                steps = max(1, self._subdivisions.getValue())
                step = length / steps

                u = 0
                for i in range(steps + 1):
                    d1 = Topo.EdgeTool.d1(edge, u)
                    d1_dir = d1.v1.normalized()
                    d1_pnt = d1.p

                    m_x = Geom.Vec(d1_dir.x(), d1_dir.y(), d1_dir.z())
                    m_y = Geom.Vec(-d1_dir.y(), d1_dir.x(), 0.).normalized()
                    m_z = m_x.crossed(m_y).normalized()
                    m = Geom.Mat(m_x.xyz(), m_y.xyz(), m_z.xyz())

                    t = Geom.Trsf(m, d1_pnt.xyz(), 1.)

                    geo = lx.RightCircularCylinder.createIn(self.getDocument())
                    geo.setHeight(1.)
```

Get rid of the BoundingBox

Please avoid words with spaces: this should be a short unique name inside the script, not a description.

The description for the user is taken by the "translatorID" (replace -1 with the corresponding "id" ... ask Helder or Philippe for this)

Please prepend a "_" to this name so we avoid conflicts with other names that may be used by Lexocad.

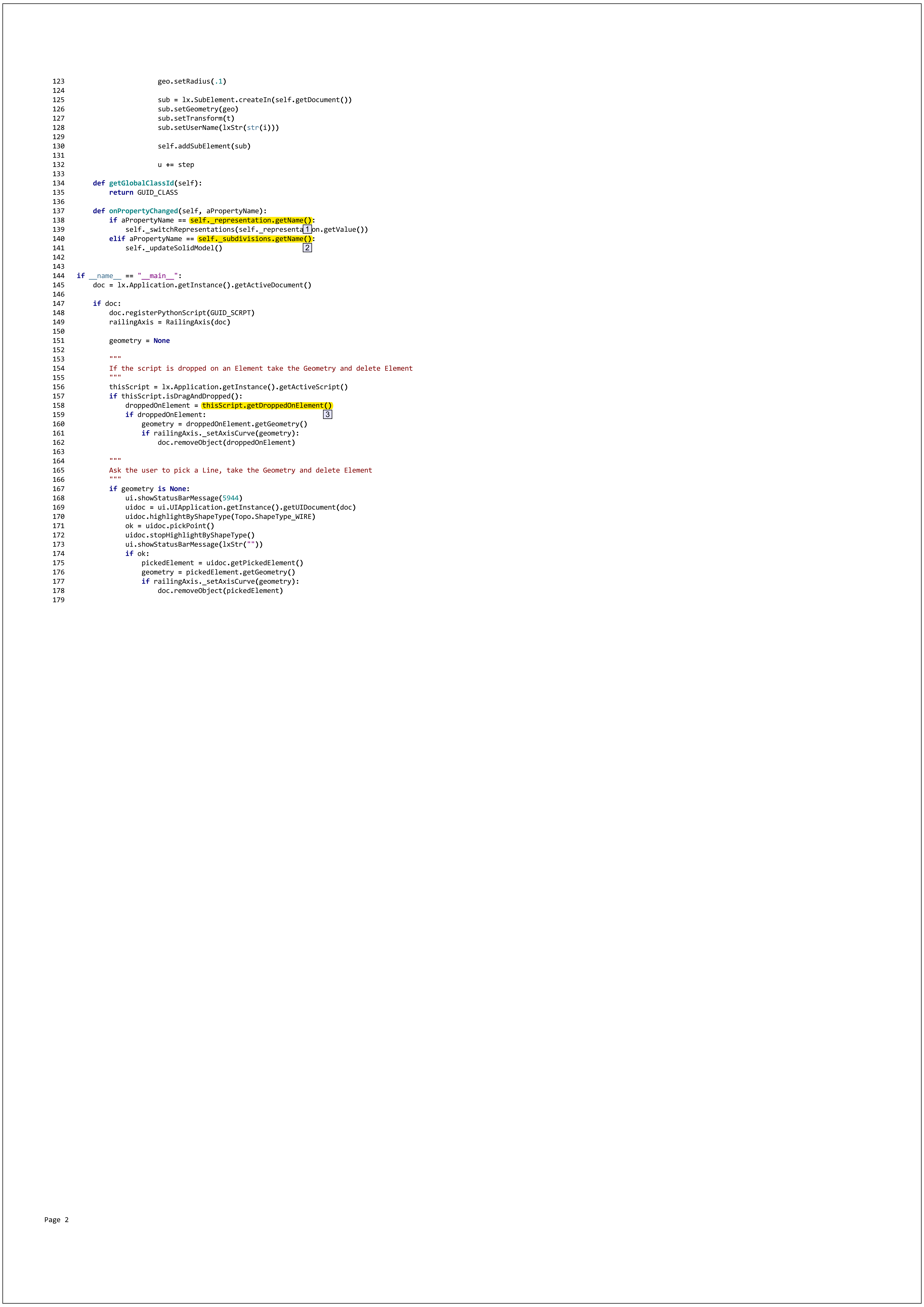
Here we store the Axis

Here we tell Lexocad to show the Axis or the SolidModel (in our case it is the MultiGeo) depending on the user choice.

Since the Axis may have been changed, we recreate the MultiGeo again.

Here we get the stored Axis

Notizen



```
                    geo.setRadius(.1)

                    sub = lx.SubElement.createIn(self.getDocument())
                    sub.setGeometry(geo)
                    sub.setTransform(t)
                    sub.setUserName(lxStr(str(i)))

                    self.addSubElement(sub)

                    u += step

    def getGlobalClassId(self):
        return GUID_CLASS

    def onPropertyChanged(self, aPropertyName):
        if aPropertyName == self._representation.getName():
            self._switchRepresentations(self._representation.getValue())
        elif aPropertyName == self._subdivisions.getName():
            self._updateSolidModel()


if __name__ == "__main__":
    doc = lx.Application.getInstance().getActiveDocument()

    if doc:
        doc.registerPythonScript(GUID_SCRPT)
        railingAxis = RailingAxis(doc)

        geometry = None

        """
        If the script is dropped on an Element take the Geometry and delete Element
        """
        thisScript = lx.Application.getInstance().getActiveScript()
        if thisScript.isDragAndDropped():
            droppedOnElement = thisScript.getDroppedOnElement()
            if droppedOnElement:
                geometry = droppedOnElement.getGeometry()
                if railingAxis._setAxisCurve(geometry):
                    doc.removeObject(droppedOnElement)

        """
        Ask the user to pick a Line, take the Geometry and delete Element
        """
        if geometry is None:
            ui.showStatusBarMessage(5944)
            uidoc = ui.UIApplication.getInstance().getUIDocument(doc)
            uidoc.highlightByShapeType(Topo.ShapeType_WIRE)
            ok = uidoc.pickPoint()
            uidoc.stopHighlightByShapeType()
            ui.showStatusBarMessage(lxStr(""))
            if ok:
                pickedElement = uidoc.getPickedElement()
                geometry = pickedElement.getGeometry()
                if railingAxis._setAxisCurve(geometry):
                    doc.removeObject(pickedElement)
```

1 You don't need to store/remember the name of the property that may be changing.

2 The property's name can be queried with .getName()

3 Here we can catch the Element where the Script has been Dropped onto.

Notizen