Summer Fellowship Report

On

Xcos-On-Cloud

Submitted by

Avinash Agarwal
Swanand Pande

Under the guidance of

Prof.Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

July 2, 2018

Acknowledgment

We would like to thank everyone who have been instrumental in helping us achieve the desired outcome. We would like to thank Prof.Kannan Moudgalya and IITB for providing us with this opportunity to work on this project and also having faith in our abilities. We would like to thank our mentors Ms. Dipti Ghosalkar and Ms. Rinki Nag for being a constant support and giving us a broader picture of the whole scenario. Their constant endeavor and guidance has helped us to complete the project within the stipulated deadline. We would also like to thank Ms. Komal Solanki for providing us the administrative help and making our working environment comfortable. Last but not the least, we would like to thank our fellow interns for helping us out with all the problems that we faced and making our internship experience an enjoyable one.

Contents

Chapter 1

Introduction

Xcos is a graphical editor to design hybrid dynamical systems models. Models can be designed, loaded, saved, compiled and simulated.Ergonomic and efficient solution for industrial and academics needs, Xcos provides functionalities for modeling of mechanical systems, hydraulic circuits, control systems, etc.
Xcos-On-Cloud ports these core functionalities to a browser only version of Xcos. Shifting Xcos to browser increases its industrial and educational use as user will be able to use it anytime from anywhere without going into the trouble of installing a software.

Chapter 2

Limitations with earlier version

i. Parameters of LOOKUP_f and CURV_f couldnt be changed.

ii. After any parameter of a block in the diagram was changed, the wires connecting different blocks would disappear.

iii. Import function could only import diagram and not the values of individual blocks, i.e the import function used to initialize the block parameter values to their default value.

iv. Scifunc_block not functioning.

Chapter 3

Properties window for LOOKUP_f and CURV_f

3.1 Issue

On double clicking any block a show properties window is required to be opened for the user to change the values of different parameters. This functionality was earlier not included for LOOKUP_f and CURV_f.
The properties window for the above blocks is different than the others, in all other blocks property window contains different input fields but in case of these blocks the property window consist of the functionalities as written below.

i. A graph whose points can be dragged to new locations, and clicking anywhere inside the graph window would create a new graph point.

ii. The graph points can be saved to a coordinate.xy file locally.

iii. The graph points can be read from a file as well.

iv. The limits of the axis of graph can be changed.

v. Other features such as - Undo, Clear, Replot, Abort.

3.2 Methodology

The above was achieved by the following approach:

i. Default graph points and other data is defined under define function of LOOKUP and CURV in combined.js. This was done in accordance with the LOOKUP_f.sci/CURV_f.sci file i.e parsing scilab code to JS.

ii. Draggable graph code + read/save/clear/undo/replot options code is present inside *showGraphWindow* function inside LOOKUP_CURV.js which is called inside of *showPropertiesWindow* else block in index.js.

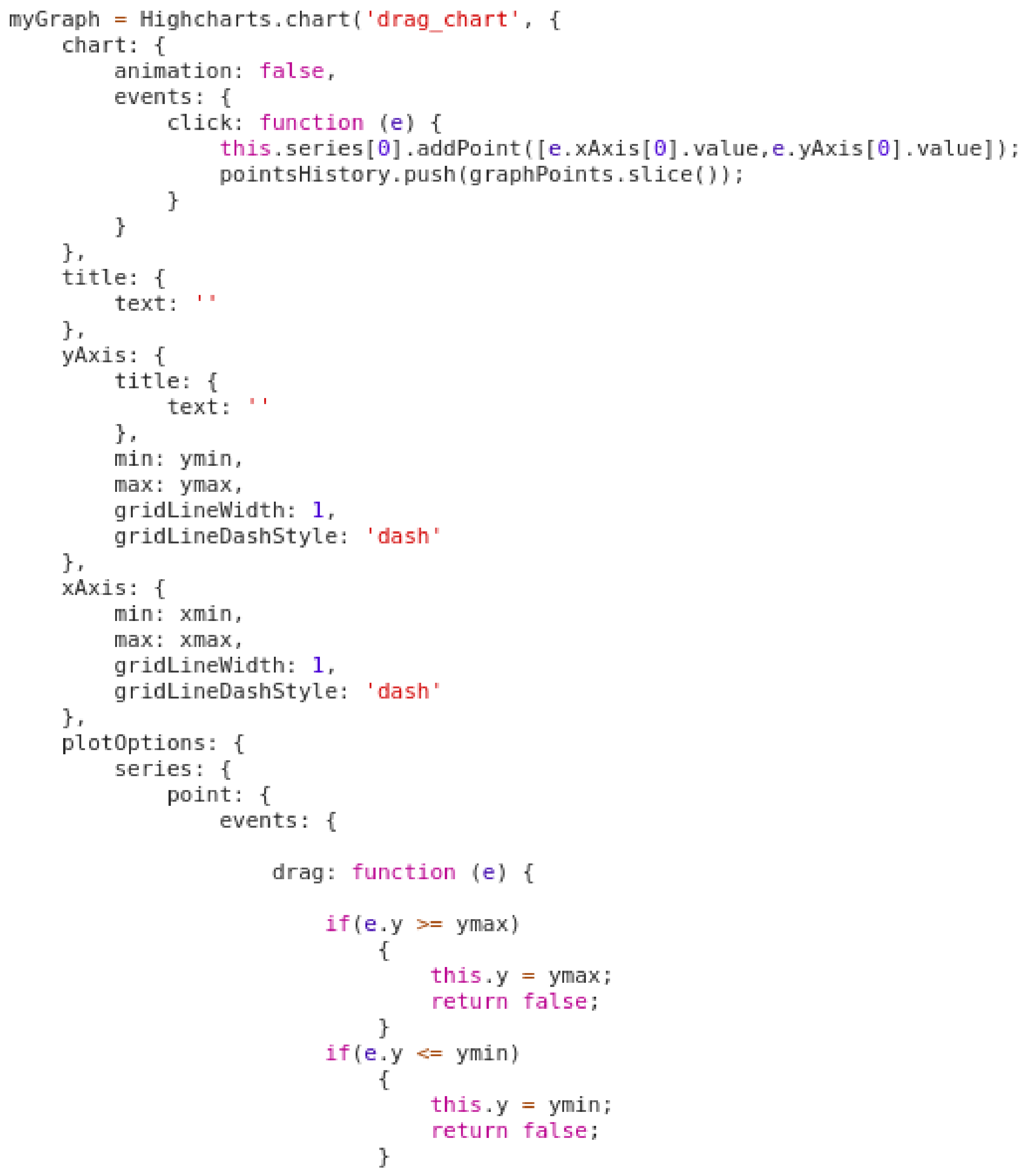
    a. Blob is used to save graph points.

b. File reader is used to read/load points.xy file from local computer.

c. Undo - In order to implement undo functionality, an array *pointsHistory* is defined to store all the previous graph states before the undo function is revoked. Once the undo button is clicked the array becomes empty.

d. Draggable graph- with the use of highcharts and event listeners.

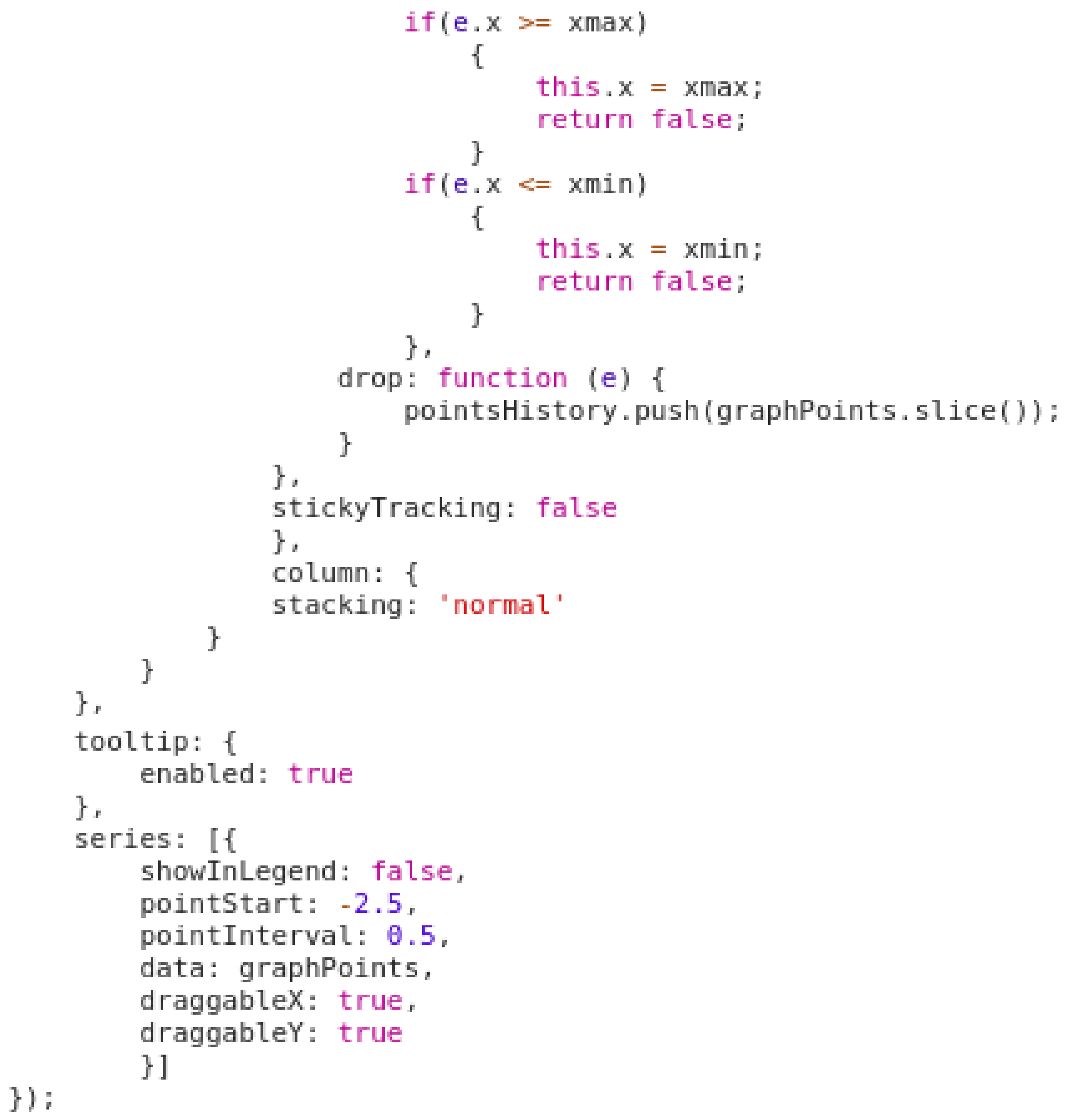
iii. Graph with draggable Points:



```
myGraph = Highcharts.chart('drag_chart', {
    chart: {
        animation: false,
        events: {
            click: function (e) {
                this.series[0].addPoint([e.xAxis[0].value,e.yAxis[0].value]);
                pointsHistory.push(graphPoints.slice());
            }
        }
    },
    title: {
        text: ''
    },
    yAxis: {
        title: {
            text: ''
        },
        min: ymin,
        max: ymax,
        gridLineWidth: 1,
        gridLineDashStyle: 'dash'
    },
    xAxis: {
        min: xmin,
        max: xmax,
        gridLineWidth: 1,
        gridLineDashStyle: 'dash'
    },
    plotOptions: {
        series: {
            point: {
                events: {

                    drag: function (e) {

                        if(e.y >= ymax)
                            {
                                this.y = ymax;
                                return false;
                            }
                        if(e.y <= ymin)
                            {
                                this.y = ymin;
                                return false;
                            }
```



```
                        if(e.x >= xmax)
                            {
                                this.x = xmax;
                                return false;
                            }
                        if(e.x <= xmin)
                            {
                                this.x = xmin;
                                return false;
                            }
                        },
                    drop: function (e) {
                        pointsHistory.push(graphPoints.slice());
                    }
                },
                stickyTracking: false
                },
                column: {
                stacking: 'normal'
            }
        }
    },
    tooltip: {
        enabled: true
    },
    series: [{
        showInLegend: false,
        pointStart: -2.5,
        pointInterval: 0.5,
        data: graphPoints,
        draggableX: true,
        draggableY: true
        }]
});
```

   a. Here, the click event listener is used for creating a new point as soon as the user clicks anywhere in the chart window.
   b. The drag event is used to set limits so that a point cannot be dragged beyond the axis limits.
   c. *draggableX: true* and *draggableY: true*, implements the drag functionality. This is an inbuilt function which is present in draggable_points.js.
   d. *pointsHistory* array is used to store previous graph states.

iv. At last a function called update model is called where a XML is created containing information about the block with its changed parameters. Inside this update function (in index.js) a set function is called to which the updated values are send. This is also done in accordance with concerned .sci files.

v. Functions created:

   a. *showGraphWindow* - To show the graph window popup when LOOKUP_f/CURV_f is double clicked.

b. create_draggable_graph - This function is responsible for creating a graph with draggable points.

c. *objToArrayList* - In highcharts, if a point is dragged its coordinates get converted to an object. This function is responsible for for converting all those points that have been converted to object because of dragging back to an array.

Chapter 4

Connecting Wires

4.1 Issue

After the changing the parameters of any block, the input and output wires connecting different blocks were disappearing.
In order to solve this problem, two problems were needed to be solved.

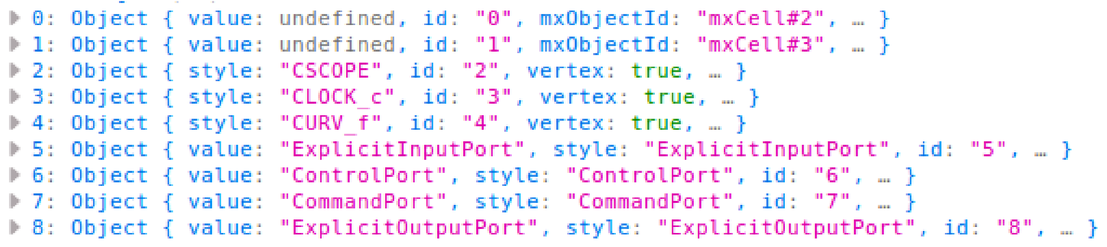
i. First, the problem of IDs of block being changed when parameters of the blocks were changed.

ii. Secondly, the *Link* nodes responsible for joining the block also disappeared from the XML when the parameters were changed. (*Link* nodes contains two necessary values, the IDs of source port of a block and target port of a block).

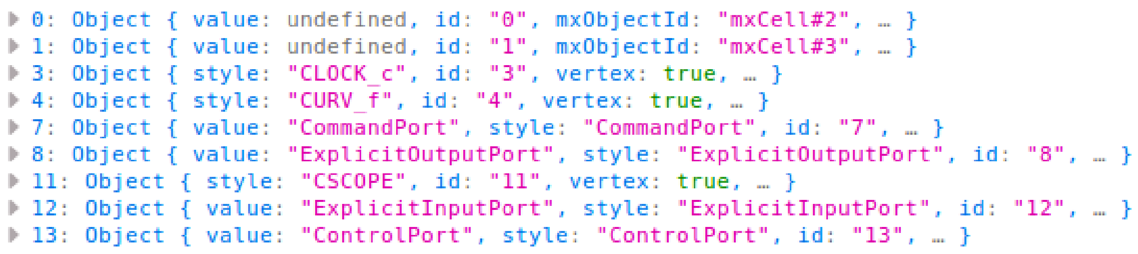
So in order to solve the first problem, We used a reference Model which stored the Id and names of all the previous node (*refereneModelProps* present inside *graph.dblClick* function). As soon as the set function was called and new nodes were created with new id, those nodes were moved back to their previous positions and assigned them there previous IDs. Like this the IDs of all blocks were retained.
(Note :- Here the IDs of all nodes remains constant except the IDs of *Link* nodes, because the IDs of *Link* nodes are independent i.e their IDs doesnt influence the output in any form).

4.2 Reason for IDs being changed

When the property of a block was changed the *node* of the block along with its input, output and command ports nodes were also deleted from the model and new nodes were created at the end of the model, it is because of this reason that the IDs of the blocks were changed and the *Explicit link* node (containing the information about old IDs) couldn't form links between the blocks anymore.
The change on IDs after properties of CSCOPE is changed is shown below.



```
▶ 0: Object { value: undefined, id: "0", mxObjectId: "mxCell#2", … }
▶ 1: Object { value: undefined, id: "1", mxObjectId: "mxCell#3", … }
▶ 2: Object { style: "CSCOPE", id: "2", vertex: true, … }
▶ 3: Object { style: "CLOCK_c", id: "3", vertex: true, … }
▶ 4: Object { style: "CURV_f", id: "4", vertex: true, … }
▶ 5: Object { value: "ExplicitInputPort", style: "ExplicitInputPort", id: "5", … }
▶ 6: Object { value: "ControlPort", style: "ControlPort", id: "6", … }
▶ 7: Object { value: "CommandPort", style: "CommandPort", id: "7", … }
▶ 8: Object { value: "ExplicitOutputPort", style: "ExplicitOutputPort", id: "8", … }
```



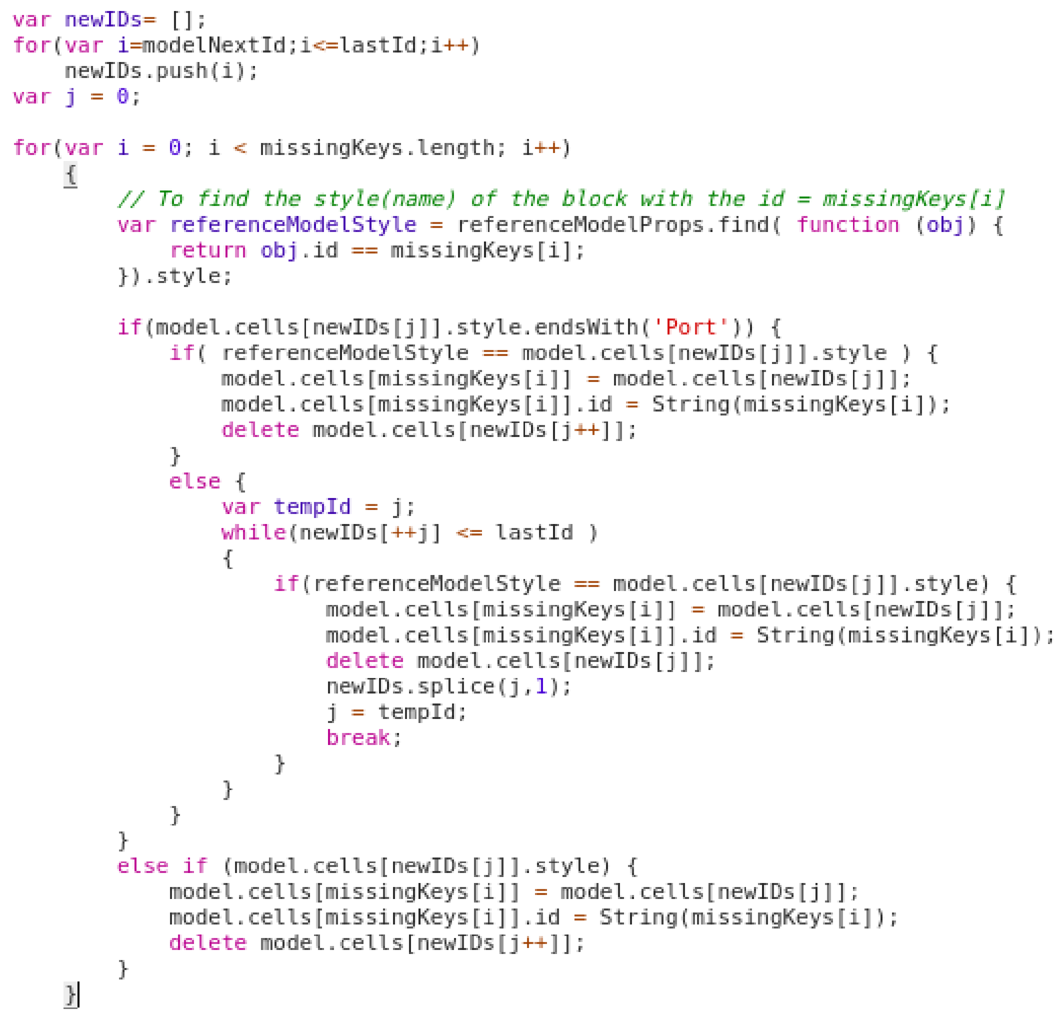
```
▶ 0: Object { value: undefined, id: "0", mxObjectId: "mxCell#2", … }
▶ 1: Object { value: undefined, id: "1", mxObjectId: "mxCell#3", … }
▶ 3: Object { style: "CLOCK_c", id: "3", vertex: true, … }
▶ 4: Object { style: "CURV_f", id: "4", vertex: true, … }
▶ 7: Object { value: "CommandPort", style: "CommandPort", id: "7", … }
▶ 8: Object { value: "ExplicitOutputPort", style: "ExplicitOutputPort", id: "8", … }
▶ 11: Object { style: "CSCOPE", id: "11", vertex: true, … }
▶ 12: Object { value: "ExplicitInputPort", style: "ExplicitInputPort", id: "12", … }
▶ 13: Object { value: "ControlPort", style: "ControlPort", id: "13", … }
```

4.3 Methodology

referenceModelProps array stores the IDs and names of all nodes and is updated every time a block is double clicked so that, the previous model is used for comparing the new model created(on changing the parameters) to retain the IDs.
modelNextId stores the ID of the new node to be created.

```
var referenceModel = graph.getModel();
var element_count = 0;
for (var e in referenceModel.cells)
{
    if(referenceModel.cells.hasOwnProperty(e))
        if(referenceModel.cells[e].style)
        {
            referenceModelProps[element_count++] = { id: e, style: referenceModel.cells[e].style };
        }
}
modelNextId = referenceModel.nextId;
```

An array *missingKeys* is used to store the ids of the missing keys in the new model by comparing old model with the new model.
Then the following code is used to shift the new nodes with new ids back to their previous position and is given their old id.
As shown below, *newIDs* is an array which stores all the new ids created before link nodes are created.
Then in the loop we check if the node is not a port node then we shift it back to its old id and its old id is assigned to it. But if the node is a port node, then the name of the node which was earlier present at the old id is checked against all the new id when a match is found the new id is deleted from the *newIDs* array and shifted back to its old id. (This is done so that even when new ports are created the connections dont get lost).



```
var newIDs= [];
for(var i=modelNextId;i<=lastId;i++)
    newIDs.push(i);
var j = 0;

for(var i = 0; i < missingKeys.length; i++)
    {
        // To find the style(name) of the block with the id = missingKeys[i]
        var referenceModelStyle = referenceModelProps.find( function (obj) {
            return obj.id == missingKeys[i];
        }).style;

        if(model.cells[newIDs[j]].style.endsWith('Port')) {
            if( referenceModelStyle == model.cells[newIDs[j]].style ) {
                model.cells[missingKeys[i]] = model.cells[newIDs[j]];
                model.cells[missingKeys[i]].id = String(missingKeys[i]);
                delete model.cells[newIDs[j++]];
            }
            else {
                var tempId = j;
                while(newIDs[++j] <= lastId )
                {
                    if(referenceModelStyle == model.cells[newIDs[j]].style) {
                        model.cells[missingKeys[i]] = model.cells[newIDs[j]];
                        model.cells[missingKeys[i]].id = String(missingKeys[i]);
                        delete model.cells[newIDs[j]];
                        newIDs.splice(j,1);
                        j = tempId;
                        break;
                    }
                }
            }
        }
        else if (model.cells[newIDs[j]].style) {
            model.cells[missingKeys[i]] = model.cells[newIDs[j]];
            model.cells[missingKeys[i]].id = String(missingKeys[i]);
            delete model.cells[newIDs[j++]];
        }
    }
```

For solving the second issue, once the IDs were constant, We used the XML of the previous graphs state (as it contained the information about the *link* nodes) to create the *link* nodes again in the new XML.The code below is responsible for creating the links.

```
while(currentNode!=null)
{
    var curNodeName = currentNode.nodeName;
    if(curNodeName.endsWith('Link'))
       {
           var pointsArray = [];
           var newSourceCell = graph.getModel().getCell(currentNode.getAttribute('source'));
           var newTargetCell = graph.getModel().getCell(currentNode.getAttribute('target'));

            if(newSourceCell.getEdgeCount() <=0 && newTargetCell.getEdgeCount()<=0) {

               var childNode = currentNode.firstChild;
                    if (childNode != null) {
                        if (childNode.nodeName == 'mxGeometry') {
                            var tempNode = childNode.firstChild;
                            if (tempNode != null) {
                                if (tempNode.nodeName == 'mxPoint') {
                                    pointsArray.push(new mxPoint(tempNode.getAttribute('x'), tempNode.getAttribute('y')));
                                } else {
                                    if (tempNode.nodeName == 'Array') {
                                        var mxPointNode = tempNode.firstChild;
                                        while (mxPointNode != null) {
                                            pointsArray.push(new mxPoint(mxPointNode.getAttribute('x'), mxPointNode.getAttribute('y')));
                                            mxPointNode = mxPointNode.nextSibling;
                                        }
                                    }
                                }
                            }
                        }
                    }

                createEdgeObject(graph, newSourceCell, newTargetCell, null);
            }
       }
    currentNode=currentNode.nextSibling;
}
```

Chapter 5

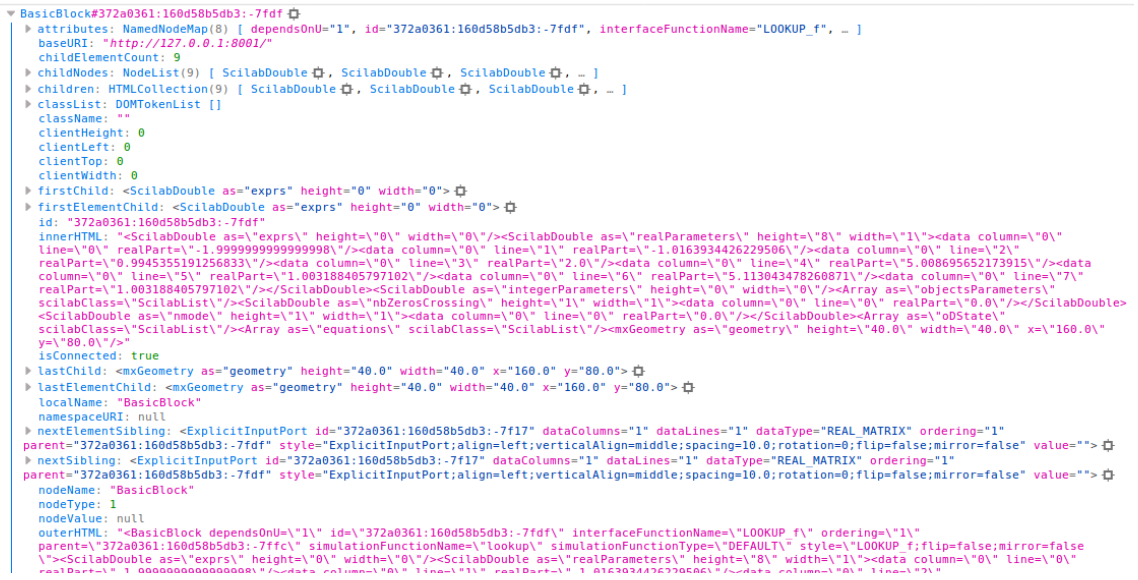
Importing Block Parameters Values

5.1 Issue

Import function could only import diagram and not the values of individual blocks.

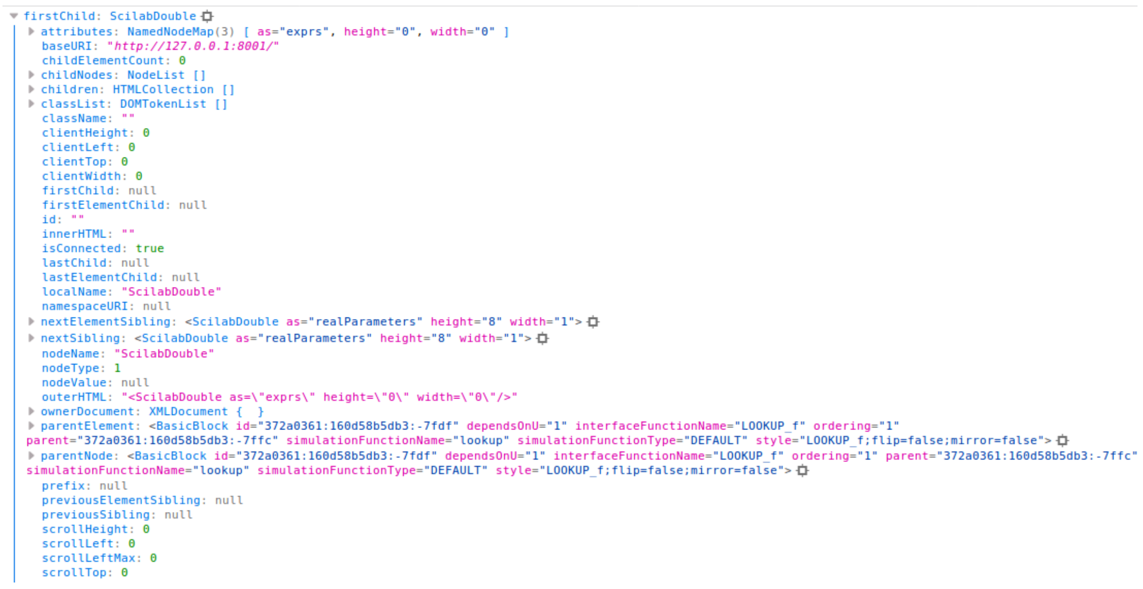
5.2 Methodology

Reading the parameter values of the block from the XML loaded from import and calling respective set function of the blocks with those values . The main challenge here was parsing the XML to find the values. Parsing of XML can be done in the following manner:
The import function first takes the scilab compatible XML and converts it to mxgraph compatible XML.The mxgraph compatible XML object looks as shown below -

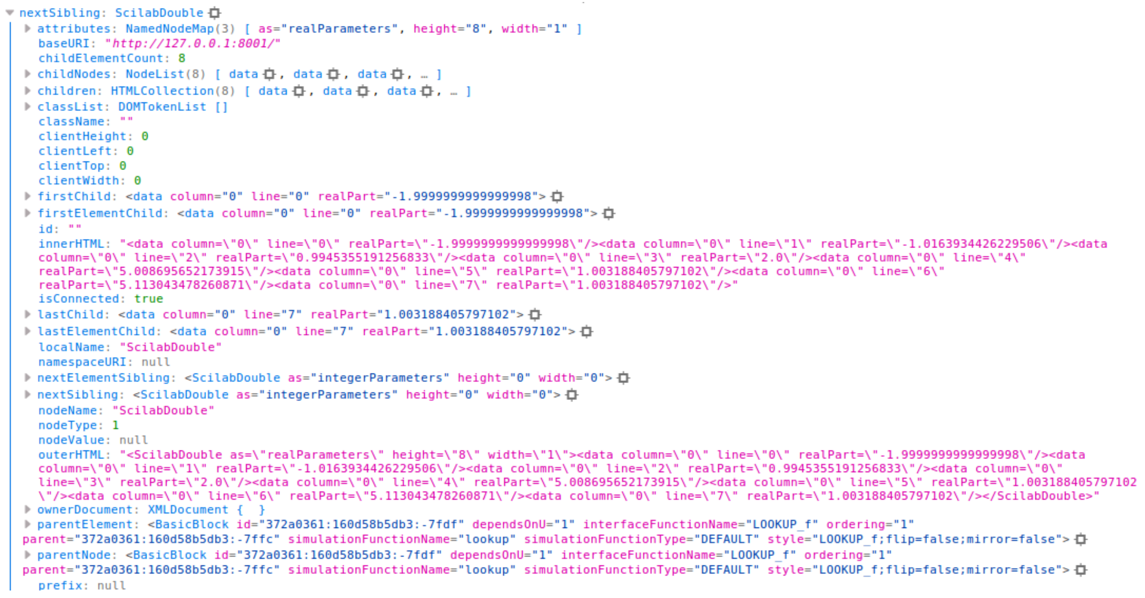


```
▾ BasicBlock#372a0361:160d58b5db3:-7fdf
  ▸ attributes: NamedNodeMap(8) [ dependsOnU="1", id="372a0361:160d58b5db3:-7fdf", interfaceFunctionName="LOOKUP_f", … ]
    baseURI: "http://127.0.0.1:8001/"
    childElementCount: 9
  ▸ childNodes: NodeList(9) [ ScilabDouble, ScilabDouble, ScilabDouble, … ]
  ▸ children: HTMLCollection(9) [ ScilabDouble, ScilabDouble, ScilabDouble, … ]
  ▸ classList: DOMTokenList []
    className: ""
    clientHeight: 0
    clientLeft: 0
    clientTop: 0
    clientWidth: 0
  ▸ firstChild: <ScilabDouble as="exprs" height="0" width="0">
  ▸ firstElementChild: <ScilabDouble as="exprs" height="0" width="0">
    id: "372a0361:160d58b5db3:-7fdf"
    innerHTML: "<ScilabDouble as=\"exprs\" height=\"0\" width=\"0\"/><ScilabDouble as=\"realParameters\" height=\"8\" width=\"1\"><data column=\"0\"
    line=\"0\" realPart=\"-1.9999999999999998\"/><data column=\"0\" line=\"1\" realPart=\"-1.0163934426229506\"/><data column=\"0\" line=\"2\"
    realPart=\"0.9945355191256833\"/><data column=\"0\" line=\"3\" realPart=\"2.0\"/><data column=\"0\" line=\"4\" realPart=\"5.008695652173915\"/><data
    column=\"0\" line=\"5\" realPart=\"1.003188405797102\"/><data column=\"0\" line=\"6\" realPart=\"5.113043478260871\"/><data column=\"0\" line=\"7\"
    realPart=\"1.003188405797102\"/></ScilabDouble><ScilabDouble as=\"integerParameters\" height=\"0\" width=\"0\"/><Array as=\"objectsParameters\"
    scilabClass=\"ScilabList\"/><ScilabDouble as=\"nbZerosCrossing\" height=\"1\" width=\"1\"><data column=\"0\" line=\"0\" realPart=\"0.0\"/></ScilabDouble>
    <ScilabDouble as=\"nmode\" height=\"1\" width=\"1\"><data column=\"0\" line=\"0\" realPart=\"0.0\"/></ScilabDouble><Array as=\"oDState\"
    scilabClass=\"ScilabList\"/><Array as=\"equations\" scilabClass=\"ScilabList\"/><mxGeometry as=\"geometry\" height=\"40.0\" width=\"40.0\" x=\"160.0\"
    y=\"80.0\"/>"
    isConnected: true
  ▸ lastChild: <mxGeometry as="geometry" height="40.0" width="40.0" x="160.0" y="80.0">
  ▸ lastElementChild: <mxGeometry as="geometry" height="40.0" width="40.0" x="160.0" y="80.0">
    localName: "BasicBlock"
    namespaceURI: null
  ▸ nextElementSibling: <ExplicitInputPort id="372a0361:160d58b5db3:-7f17" dataColumns="1" dataLines="1" dataType="REAL_MATRIX" ordering="1"
  parent="372a0361:160d58b5db3:-7fdf" style="ExplicitInputPort;align=left;verticalAlign=middle;spacing=10.0;rotation=0;flip=false;mirror=false" value="">
  ▸ nextSibling: <ExplicitInputPort id="372a0361:160d58b5db3:-7f17" dataColumns="1" dataLines="1" dataType="REAL_MATRIX" ordering="1"
  parent="372a0361:160d58b5db3:-7fdf" style="ExplicitInputPort;align=left;verticalAlign=middle;spacing=10.0;rotation=0;flip=false;mirror=false" value="">
    nodeName: "BasicBlock"
    nodeType: 1
    nodeValue: null
    outerHTML: "<BasicBlock dependsOnU=\"1\" id=\"372a0361:160d58b5db3:-7fdf\" interfaceFunctionName=\"LOOKUP_f\" ordering=\"1\"
    parent=\"372a0361:160d58b5db3:-7ffc\" simulationFunctionName=\"lookup\" simulationFunctionType=\"DEFAULT\" style=\"LOOKUP_f;flip=false;mirror=false
    \"><ScilabDouble as=\"exprs\" height=\"0\" width=\"0\"/><ScilabDouble as=\"realParameters\" height=\"8\" width=\"1\"><data column=\"0\" line=\"0\"
```

In this *datatype firstChild* contains information about the next nested node and *nextSibling* contains information about the node following the parent node.



```
▼ firstChild: ScilabDouble
  ▶ attributes: NamedNodeMap(3) [ as="exprs", height="0", width="0" ]
    baseURI: "http://127.0.0.1:8001/"
    childElementCount: 0
  ▶ childNodes: NodeList []
  ▶ children: HTMLCollection []
  ▶ classList: DOMTokenList []
    className: ""
    clientHeight: 0
    clientLeft: 0
    clientTop: 0
    clientWidth: 0
    firstChild: null
    firstElementChild: null
    id: ""
    innerHTML: ""
    isConnected: true
    lastChild: null
    lastElementChild: null
    localName: "ScilabDouble"
    namespaceURI: null
  ▶ nextElementSibling: <ScilabDouble as="realParameters" height="8" width="1">
  ▶ nextSibling: <ScilabDouble as="realParameters" height="8" width="1">
    nodeName: "ScilabDouble"
    nodeType: 1
    nodeValue: null
    outerHTML: "<ScilabDouble as=\"exprs\" height=\"0\" width=\"0\"/>"
  ▶ ownerDocument: XMLDocument {  }
  ▶ parentElement: <BasicBlock id="372a0361:160d58b5db3:-7fdf" dependsOnU="1" interfaceFunctionName="LOOKUP_f" ordering="1"
  parent="372a0361:160d58b5db3:-7ffc" simulationFunctionName="lookup" simulationFunctionType="DEFAULT" style="LOOKUP_f;flip=false;mirror=false">
  ▶ parentNode: <BasicBlock id="372a0361:160d58b5db3:-7fdf" dependsOnU="1" interfaceFunctionName="LOOKUP_f" ordering="1" parent="372a0361:160d58b5db3:-7ffc"
  simulationFunctionName="lookup" simulationFunctionType="DEFAULT" style="LOOKUP_f;flip=false;mirror=false">
    prefix: null
    previousElementSibling: null
    previousSibling: null
    scrollHeight: 0
    scrollLeft: 0
    scrollLeftMax: 0
    scrollTop: 0
```



```
▼ nextSibling: ScilabDouble
  ▶ attributes: NamedNodeMap(3) [ as="realParameters", height="8", width="1" ]
    baseURI: "http://127.0.0.1:8001/"
    childElementCount: 8
  ▶ childNodes: NodeList(8) [ data, data, data, … ]
  ▶ children: HTMLCollection(8) [ data, data, data, … ]
  ▶ classList: DOMTokenList []
    className: ""
    clientHeight: 0
    clientLeft: 0
    clientTop: 0
    clientWidth: 0
  ▶ firstChild: <data column="0" line="0" realPart="-1.9999999999999998">
  ▶ firstElementChild: <data column="0" line="0" realPart="-1.9999999999999998">
    id: ""
    innerHTML: "<data column=\"0\" line=\"0\" realPart=\"-1.9999999999999998\"/><data column=\"0\" line=\"1\" realPart=\"-1.0163934426229506\"/><data
    column=\"0\" line=\"2\" realPart=\"0.9945355191256833\"/><data column=\"0\" line=\"3\" realPart=\"2.0\"/><data column=\"0\" line=\"4\"
    realPart=\"5.008695652173915\"/><data column=\"0\" line=\"5\" realPart=\"1.003188405797102\"/><data column=\"0\" line=\"6\"
    realPart=\"5.113043478260871\"/><data column=\"0\" line=\"7\" realPart=\"1.003188405797102\"/>"
    isConnected: true
  ▶ lastChild: <data column="0" line="7" realPart="1.003188405797102">
  ▶ lastElementChild: <data column="0" line="7" realPart="1.003188405797102">
    localName: "ScilabDouble"
    namespaceURI: null
  ▶ nextElementSibling: <ScilabDouble as="integerParameters" height="0" width="0">
  ▶ nextSibling: <ScilabDouble as="integerParameters" height="0" width="0">
    nodeName: "ScilabDouble"
    nodeType: 1
    nodeValue: null
    outerHTML: "<ScilabDouble as=\"realParameters\" height=\"8\" width=\"1\"><data column=\"0\" line=\"0\" realPart=\"-1.9999999999999998\"/><data
    column=\"0\" line=\"1\" realPart=\"-1.0163934426229506\"/><data column=\"0\" line=\"2\" realPart=\"0.9945355191256833\"/><data column=\"0\"
    line=\"3\" realPart=\"2.0\"/><data column=\"0\" line=\"4\" realPart=\"5.008695652173915\"/><data column=\"0\" line=\"5\" realPart=\"1.003188405797102
    \"/><data column=\"0\" line=\"6\" realPart=\"5.113043478260871\"/><data column=\"0\" line=\"7\" realPart=\"1.003188405797102\"/></ScilabDouble>"
  ▶ ownerDocument: XMLDocument {  }
  ▶ parentElement: <BasicBlock id="372a0361:160d58b5db3:-7fdf" dependsOnU="1" interfaceFunctionName="LOOKUP_f" ordering="1"
  parent="372a0361:160d58b5db3:-7ffc" simulationFunctionName="lookup" simulationFunctionType="DEFAULT" style="LOOKUP_f;flip=false;mirror=false">
  ▶ parentNode: <BasicBlock id="372a0361:160d58b5db3:-7fdf" dependsOnU="1" interfaceFunctionName="LOOKUP_f" ordering="1"
  parent="372a0361:160d58b5db3:-7ffc" simulationFunctionName="lookup" simulationFunctionType="DEFAULT" style="LOOKUP_f;flip=false;mirror=false">
    prefix: null
```

var details = codec.decode(node); Is used to return the attributes of the node mentioned.
The values to be imported from the XML is contained either inside of exprs node or are randomly placed inside the XML. We accessed those node with the help of firstChild and nextSibling datatype of the mxgraph XML object and used codec.decode(node) to extract the required value.

5.3 Example

While importing CMSCOPE block the XML looks like as shown below.

```
<BasicBlock dependsOnU="1" id="-7bd6c0a3:16426fbaed4:-7fd5" interfaceFunctionName="CMSCOPE" parent="-7bd6c0a3:16426fbaed4:-7ffc"
 simulationFunctionName="cmscope" simulationFunctionType="C_OR_FORTRAN" style="CMSCOPE">
  <ScilabString as="exprs" height="11" width="1">
     <data column="0" line="0" value="1 1" />
     <data column="0" line="1" value="1 3 5 7 9 11 13 15" />
     <data column="0" line="2" value="-1" />
     <data column="0" line="3" value="[]" />
     <data column="0" line="4" value="[]" />
     <data column="0" line="5" value="-1 -5" />
     <data column="0" line="6" value="1 5" />
     <data column="0" line="7" value="30 30" />
     <data column="0" line="8" value="30" />
     <data column="0" line="9" value="0" />
     <data column="0" line="10" value="" />
  </ScilabString>
  <ScilabDouble as="realParameters" height="7" width="1">
     <data column="0" line="0" realPart="0.0" />
     <data column="0" line="1" realPart="30.0" />
     <data column="0" line="2" realPart="30.0" />
     <data column="0" line="3" realPart="-1.0" />
     <data column="0" line="4" realPart="1.0" />
     <data column="0" line="5" realPart="-5.0" />
     <data column="0" line="6" realPart="5.0" />
  </ScilabDouble>
  <ScilabDouble as="integerParameters" height="12" width="1">
     <data column="0" line="0" realPart="-1.0" />
     <data column="0" line="1" realPart="2.0" />
     <data column="0" line="2" realPart="30.0" />
     <data column="0" line="3" realPart="-1.0" />
     <data column="0" line="4" realPart="-1.0" />
     <data column="0" line="5" realPart="-1.0" />
     <data column="0" line="6" realPart="-1.0" />
     <data column="0" line="7" realPart="1.0" />
     <data column="0" line="8" realPart="1.0" />
     <data column="0" line="9" realPart="1.0" />
     <data column="0" line="10" realPart="3.0" />
     <data column="0" line="11" realPart="0.0" />
  </ScilabDouble>
```

The values that we are requires to extract is present under the exprs node, to extract all those values the following code is used -

```
var in1, clrs, win, wpos, wdim, ymin, ymax, per, N, heritance, nom;
var currentNodeCopy = currentNode;
currentNodeCopy = currentNodeCopy.firstChild;
currentNodeCopy = currentNodeCopy.firstChild;

in1 = codec.decode(currentNodeCopy).value.toString();
clrs = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
win = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
wpos = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
wdim = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
ymin = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
ymax = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
per = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
N =codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
heritance = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
nom = codec.decode(currentNodeCopy = currentNodeCopy.nextSibling).value.toString();
```

First time *currentNodeCopy.firstChild* was executed the control was passed from Basic Block to exprs, second time from *exprs* to its first child node and then all the child nodes are accessed with the help of *currentNodeCopy.nextSibling*.
All these values are stored in variables and then an object consisting of all these values is passed to the respective blocks set function.This is how we are able to import values along with the diagram.

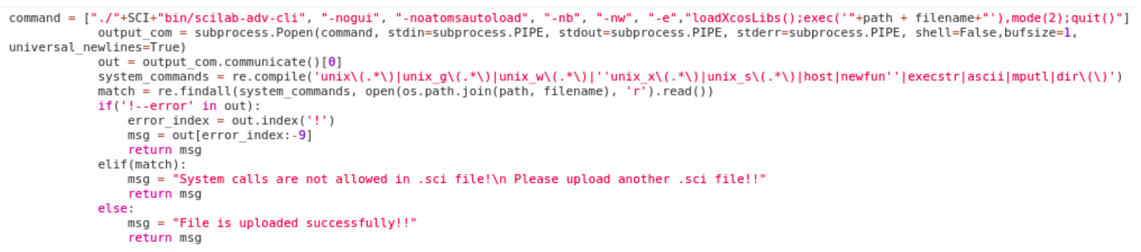
Chapter 6

Scifunc Block

6.1 Methodology

i. File browser to choose a .sci file which will be executed before executing the xcos diagram and it has to be free from two kinds of errors-

   a. System calls should not be allowed in the .sci file as it will cause problems, so the .sci file is validated to reject a file with system calls.



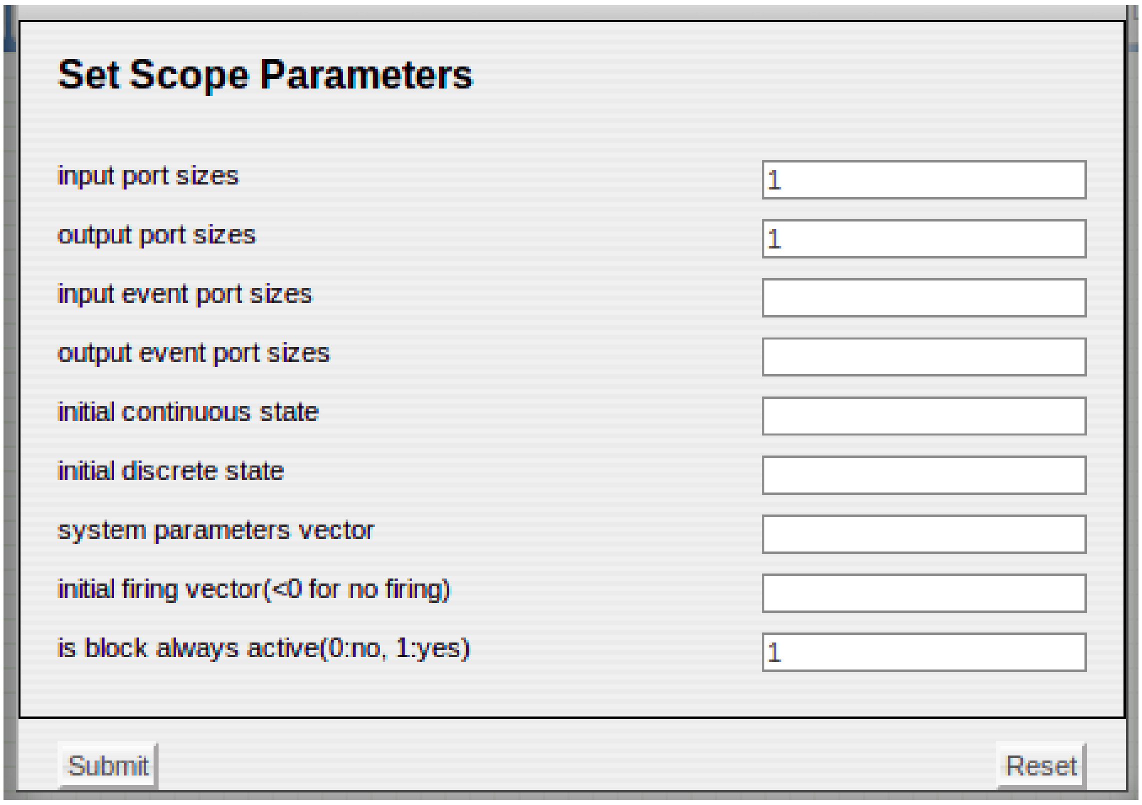
```
command = ["./"+SCI+"bin/scilab-adv-cli", "-nogui", "-noatomsautoload", "-nb", "-nw", "-e","loadXcosLibs();exec('"+path + filename+"'),mode(2);quit()"]
            output_com = subprocess.Popen(command, stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=False,bufsize=1,
universal_newlines=True)
            out = output_com.communicate()[0]
            system_commands = re.compile('unix\(.*\)|unix_g\(.*\)|unix_w\(.*\)|''unix_x\(.*\)|unix_s\(.*\)|host|newfun''|execstr|ascii|mputl|dir\(\)')
            match = re.findall(system_commands, open(os.path.join(path, filename), 'r').read())
            if('!--error' in out):
                error_index = out.index('!')
                msg = out[error_index:-9]
                return msg
            elif(match):
                msg = "System calls are not allowed in .sci file!\n Please upload another .sci file!!"
                return msg
            else:
                msg = "File is uploaded successfully!!"
                return msg
```

   b. The .sci file should not contain any undefined variable.

   c. The uploaded .sci files are stored in a folder called scifunc_files, the filenames are appended with uid and timestamp to maintain the uniqueness of files.

ii. If a proper .sci file is uploaded, then an alert box popups open saying file Uploaded Successfully. After clicking on submit, the popups open.

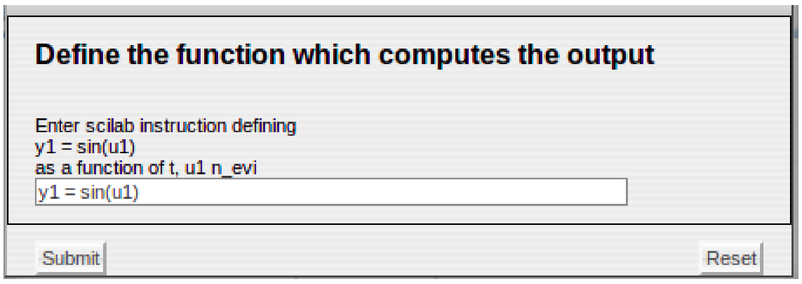
iii. In total there are 8 popups in the scifunc block, which open depending upon the parameters set.



Set Scope Parameters

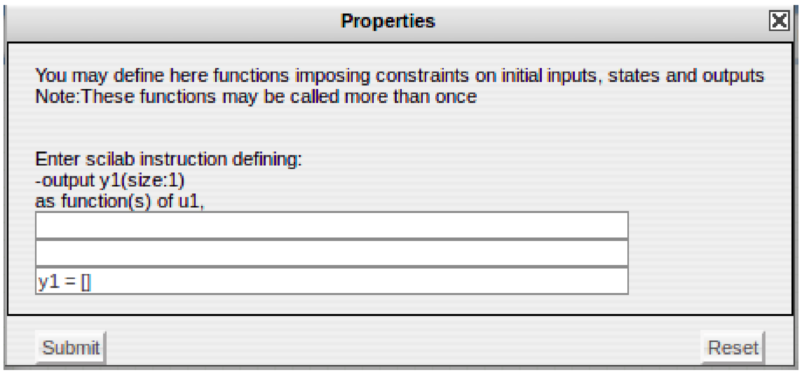
input port sizes	1
output port sizes	1
input event port sizes	
output event port sizes	
initial continuous state	
initial discrete state	
system parameters vector	
initial firing vector(<0 for no firing)	
is block always active(0:no, 1:yes)	1

iv. If the output port sizes are initialized with values like [1,1], then a popup opens at position 2 and 8 which asks to define output variables, and if the output port sizes are blank then these popups do not open.



Define the function which computes the output

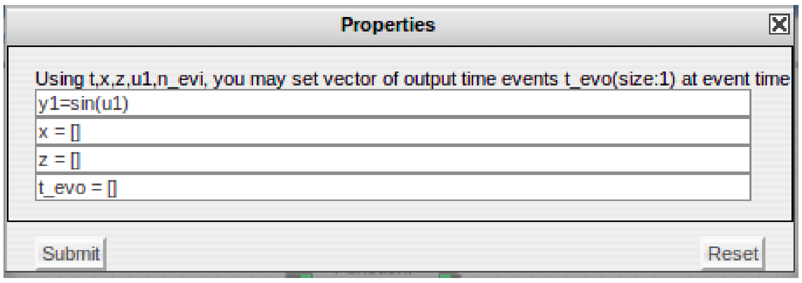
Enter scilab instruction defining
y1 = sin(u1)
as a function of t, u1 n_evi
y1 = sin(u1)



You may define here functions imposing constraints on initial inputs, states and outputs
Note:These functions may be called more than once

Enter scilab instruction defining:
-output y1(size:1)
as function(s) of u1,
y1 = []

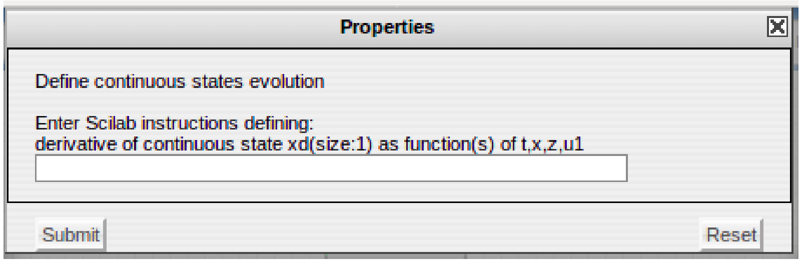
v. If input and output event port sizes are initialized, then a popup opens which asks to define vector of output time events.



Using t,x,z,u1,n_evi, you may set vector of output time events t_evo(size:1) at event time

```
y1=sin(u1)
x = []
z = []
t_evo = []
```

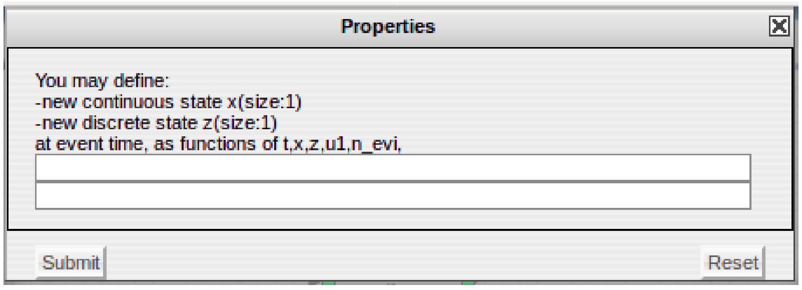
vi. If initial continuous state is initialized in first popup, then a popup opens asking user to define xd.



Define continuous states evolution

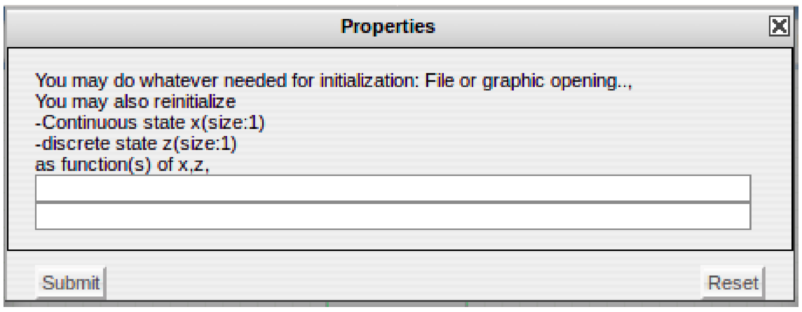
Enter Scilab instructions defining:
derivative of continuous state xd(size:1) as function(s) of t,x,z,u1

vii. If discrete continuous state is initialized in first popup, then a popup opens asking user to define a new continuous state or discrete state.

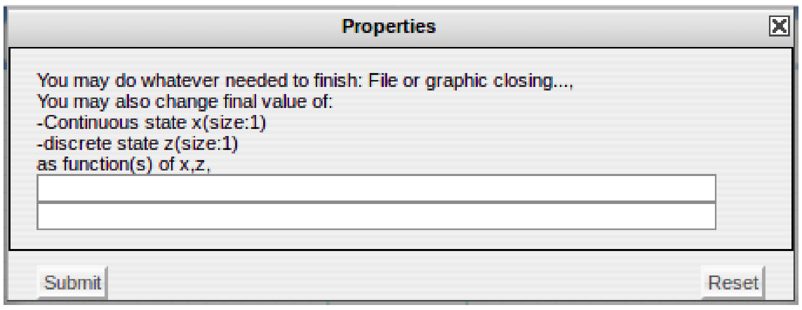


You may define:
-new continuous state x(size:1)
-new discrete state z(size:1)
at event time, as functions of t,x,z,u1,n_evi,

viii. There are two popup which always open at the end which ask user to initialize any file or graphic opening and finish any file or graphic opening.

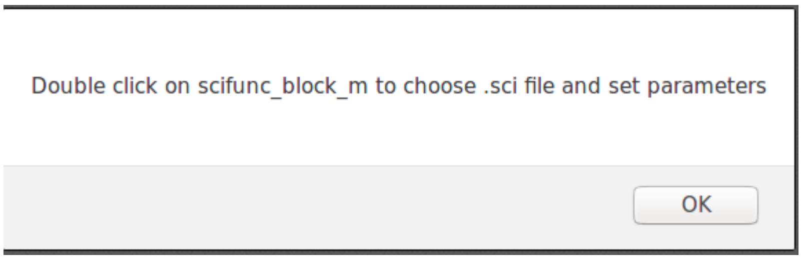


You may do whatever needed for initialization: File or graphic opening..,
You may also reinitialize
-Continuous state x(size:1)
-discrete state z(size:1)
as function(s) of x,z,

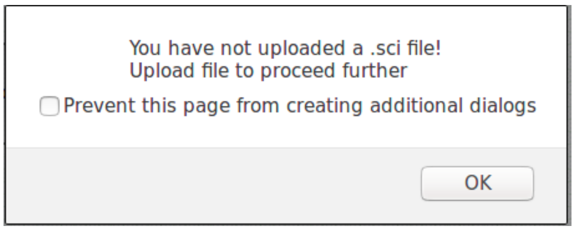

Properties

You may do whatever needed to finish: File or graphic closing...,
You may also change final value of:
-Continuous state x(size:1)
-discrete state z(size:1)
as function(s) of x,z,

ix. On importing any diagram containing scifunc block, after clicking on import, an alert is generated asking user to double click on scifunc block to upload a sci file and change parameters.

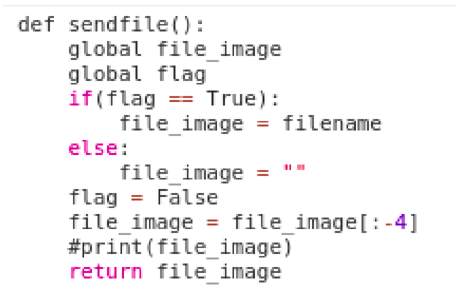

Double click on scifunc_block_m to choose .sci file and set parameters

x. Even if the user click on simulate without uploading a sci file, then an alert is generated telling user that he/she hasnt uploaded a sci file and he/she needs to do so yo proceed further.

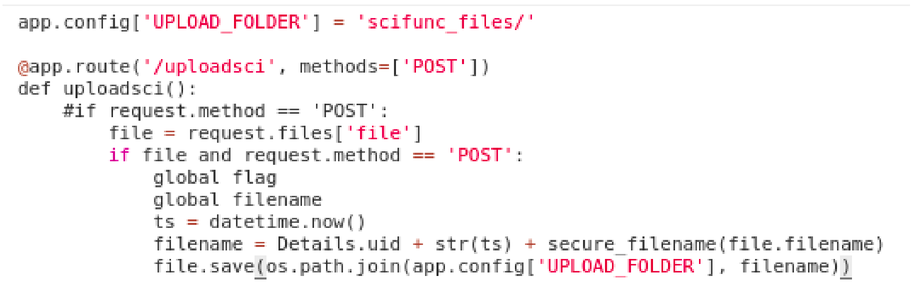

You have not uploaded a .sci file!
Upload file to proceed further

☐ Prevent this page from creating additional dialogs

xi. Initially a flag is defined as a global variable initialized to false and it becomes true only when the user uploads a file, and if it is true then only the filename is sent to do further processing and the flag is set to false again at the end.


```
def sendfile():
    global file_image
    global flag
    if(flag == True):
        file_image = filename
    else:
        file_image = ""
    flag = False
    file_image = file_image[:-4]
    #print(file_image)
    return file_image
```

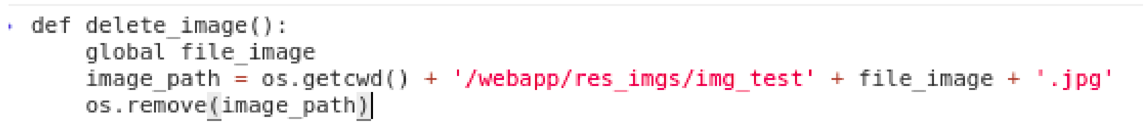
xii. A folder named scifunc_folder is created where the sci files uploaded by the user are stored. To ensure uniqueness of each file, the filename is appended with uid as well as timestamp so that always the proper file is executed.



```
app.config['UPLOAD_FOLDER'] = 'scifunc_files/'

@app.route('/uploadsci', methods=['POST'])
def uploadsci():
    #if request.method == 'POST':
        file = request.files['file']
        if file and request.method == 'POST':
            global flag
            global filename
            ts = datetime.now()
            filename = Details.uid + str(ts) + secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
```

xiii. After a diagram is imported and the sci file imported and parameters set, the image is generated in the backend and it is shown in the output window. The image is stored in the res_imgs folder which is inside the webapp folder. The image is present only for 15 seconds and is deleted afterwards as the user can do some changes in the same diagram having same file, so to avoid having any production of wrong output as the image file is stored with the same name of the file, as the filename is transferred and appended to the image name.



```
def delete_image():
    global file_image
    image_path = os.getcwd() + '/webapp/res_imgs/img_test' + file_image + '.jpg'
    os.remove(image_path)
```

Code for image deletion

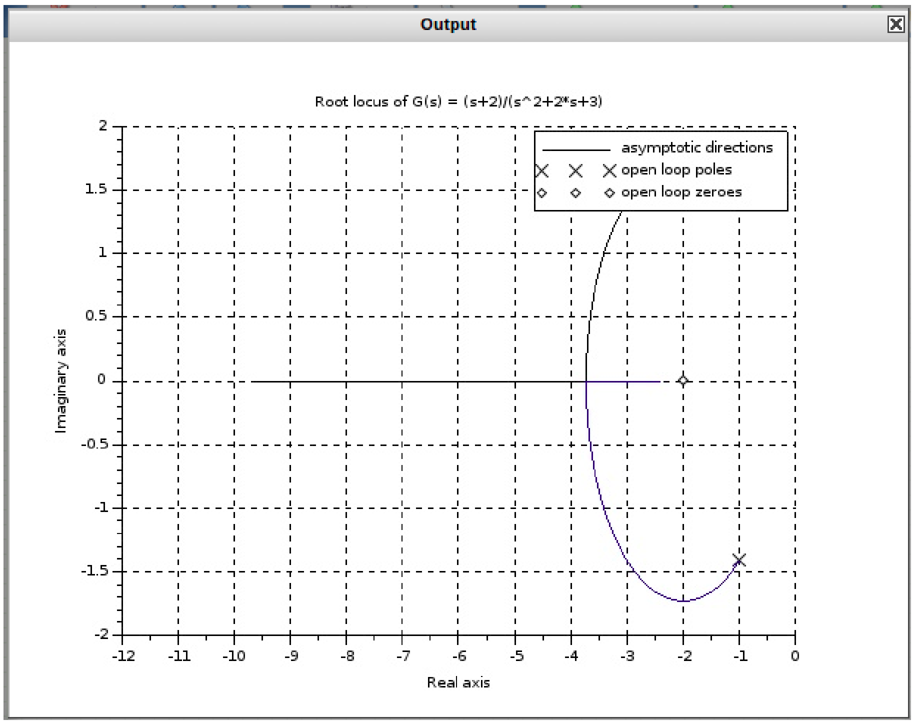
6.2 Scifunc with CSCOPE

i. In case of scifunc block with cscope block, the output should be a graph unlike in other cases where the output is an image. So for this purpose, the earlier window which opens for other blocks is retained in case of scifunc with cscope whereas in case of just scifunc block, the window is destroyed and a new window is generated which shows the output image.

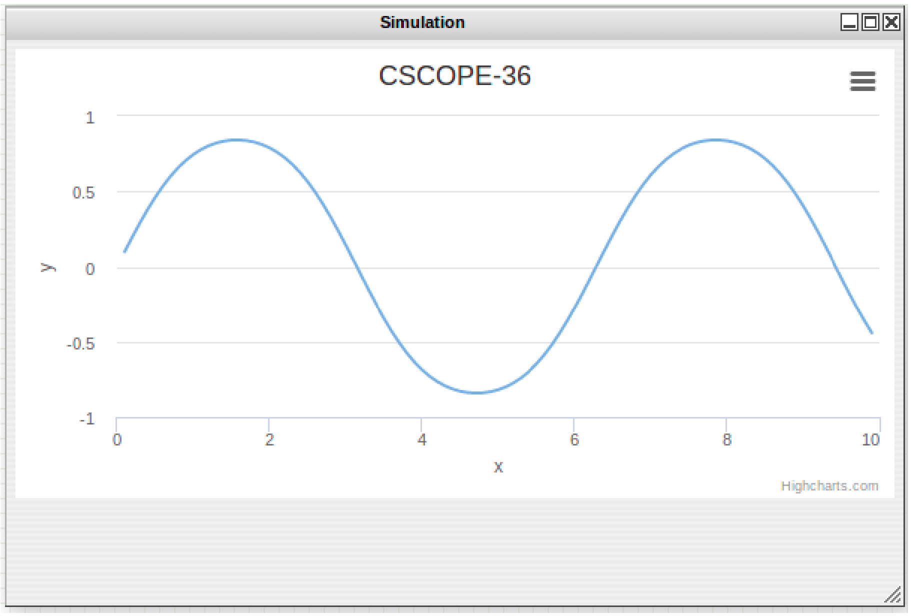
```
if(flag1 == true && flag2 == undefined){
    wnd.destroy();
    $.ajax({
        type:"post",
        url: "/requestfilename",
        async: true,
        cache: false,
        processData: false,
        contentType: false,
        success: function(name){
            console.log(name);
            if(name == ""){
                alert("You have not uploaded a .sci file!\nUpload file to proceed further");
                return false;
            }
            display_scifunc_output(name);
        },
        error: function(name){
            alert("An error occured!!");
            return false;
        }
    });
}

function display_scifunc_output(name){
    setTimeout(function(){showimage();}, 8000);
    function showimage(){
        var content = document.createElement('div');
        content.id = "image_display";
        var img_disp = document.createElement('img');
        img_disp.id = 'img';
        img_disp.src = '/res_imgs/img_test' + name + '.jpg';
        //url = img_disp.src;
        content.appendChild(img_disp);
        var wind = showModalWindow(graph, "Output", content, 610, 480);
        //window.open(url, "outputWindow", "width=700,height=500,left=400;top=300");
    }
}
```

Code for window destroying in case of only scifunc and showing output image



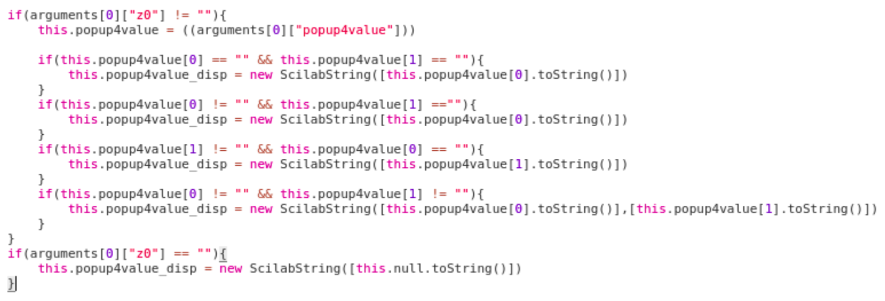
Output Image



Output graph in case of scifunc with cscope

6.3 Setting popup values

i. As more than one values are being set in each popup, there are various conditions which need to be taken care of while putting these values in the exprs.

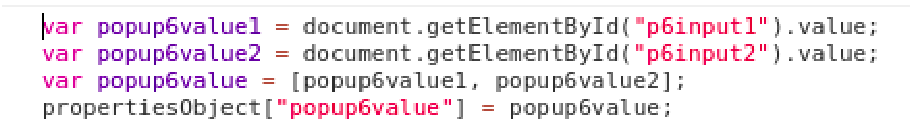


```
if(arguments[0]["z0"] != ""){
    this.popup4value = ((arguments[0]["popup4value"]))

    if(this.popup4value[0] == "" && this.popup4value[1] == ""){
        this.popup4value_disp = new ScilabString([this.popup4value[0].toString()])
    }
    if(this.popup4value[0] != "" && this.popup4value[1] ==""){
        this.popup4value_disp = new ScilabString([this.popup4value[0].toString()])
    }
    if(this.popup4value[1] != "" && this.popup4value[0] == ""){
        this.popup4value_disp = new ScilabString([this.popup4value[1].toString()])
    }
    if(this.popup4value[0] != "" && this.popup4value[1] != ""){
        this.popup4value_disp = new ScilabString([this.popup4value[0].toString()],[this.popup4value[1].toString()])
    }
}
if(arguments[0]["z0"] == ""){
    this.popup4value_disp = new ScilabString([this.null.toString()])
}
```

ii. Similar conditions are added for other popups as well while setting the value before exprs.

iii. The values are pushed in an array and the array is passed, and each value is extracted by using proper indexes.



```
var popup6value1 = document.getElementById("p6input1").value;
var popup6value2 = document.getElementById("p6input2").value;
var popup6value = [popup6value1, popup6value2];
propertiesObject["popup6value"] = popup6value;
```

Pushing values in an array and passing the propererties Object

Reference

- Highcharts Documentations - https://www.highcharts.com/docs
- Draggable Highcharts - https://github.com/highcharts/draggable-points
- mxGraph User Manual - https://jgraph.github.io/mxgraph/docs/manual.html
- AJAX Requests - https://www.w3schools.com/xml/ajax_intro.asp
- Flask - https://codehandbook.org/python-flask-jquery-ajax-post/