API Documentation

Julian Petford
Computer Science Department
University of St Andrews
St Andrews, UK
jp438@st-andrews.ac.uk

1. API CALLS

1.1 Initializing a connection

After running server.py then creating an instance of messageSender, you need to log into the server so that it can give objects you create appropriate identifiers. You do this by making both of the following calls.

1.1.1 login(username)

Logs the user in with the given username.

1.1.2 setapp(appname)

Sets the application name for the current application.

1.2 Surface Management

Surfaces act as the lowest level of the GUI tree that represents what is being shown on the walls. The outer curves of a surface dictate the transformations that occur to make the image appear unwarped on the wall. Surface creation is generally handled by the configuration program, however the following API calls exist for surface management.

1.2.1 newSurface()

This call creates a surface object and returns its surface number (surfaceNo) to be used as an identifier by future calls.

1.2.2 newSurfaceWithID(ID)

This call creates a new surface with an identifier of the user's choice and returns its surface number (surfaceNo) to be used as an identifier by future calls.

1.2.3 getSurfaceID(surfaceNo)

This call returns the ID of the surface with the given surface number.

1.2.4 setSurfaceID(surfaceNo, ID)

This call sets the ID of the surface with the given surface number.

1.2.5 getSurfaceOwner(surfaceNo)

This call returns the owner name of the surface with the given surface number.

1.2.6 getSurfaceAppDetails(surfaceNo)

This call returns the application details of the surface with the given surface number. This is returned as a tuple containing the application name and the instance number.

1.2.7 getSurfacesByID(ID)

This call returns a list of surface numbers containing all surfaces that have the queried ID.

1.2.8 getSurfacesByOwner(owner)

This call returns a list of surface numbers containing all surfaces that have the queried owner.

1.2.9 getSurfacesByAppName(name)

This call returns a list of surface numbers containing all surfaces that have the queried application name.

1.2.10 getSurfacesByAppDetails(name, instance)

This call returns a list of surface numbers containing all surfaces that have the queried application name and instance number.

1.2.11 setSurfaceEdges(surfaceNo, topPoints, bottomPoints, leftPoints, rightPoints)

Sets the edge points of the surface with the given surface number. topPoints, bottomPoints, leftPoints and rightPoints should all be lists of (x,y) coordinate tuples.

1.2.12 undefineSurface(surfaceNo)

Undefines the edge points allocated to a surface. It will no longer be rendered until the points are redefined.

1.2.13 saveDefinedSurfaces(filename)

Saves the defined surface layout in the room at the server name with the requested file name.

1.2.14 loadDefinedSurfaces(filename)

Loads a layout from the server-side file with the given name and returns a (count, surfaceslist, connections) tuple.

1.2.15 getSavedLayouts()

Returns a list of all the layouts stored on the server side.

1.2.16 deleteLayout(name)

Deletes the surface layout with the given name from the server.

1.2.17 setSurfacePixelHeight(surfaceNo, height)

Sets the height in pixels of the texture which is going to be applied to the given surface.

1.2.18 setSurfacePixelWidth(surfaceNo, width)
Sets the width in pixels of the texture which is going to be applied to the given surface.

1.2.19 clearSurface(surfaceNo)
Removes all windows and elements from the given surface.

1.2.20 rotateSurfaceTo0(surfaceNo)
Makes it so that the top left of the surface matches up with the top left of the surface's texture.

1.2.21 rotateSurfaceTo90(surfaceNo)
Makes it so that the top right of the surface matches up with the top left of the surface's texture.

1.2.22 rotateSurfaceTo180(surfaceNo)
Makes it so that the bottom right of the surface matches up with the top left of the surface's texture.

1.2.23 rotateSurfaceTo270(surfaceNo)
Makes it so that the bottom left of the surface matches up with the top left of the surface's texture.

1.2.24 mirrorSurface(surfaceNo)
mirrors the texture of the surface around the top left / bottom right diagonal.

1.2.25 connectSufaces(surfaceNo1, side1, surfaceNo2, side2)
Creates a connection between a side of one surface and a side of another surface so that cursors can move between the surfaces. Sides should be given as one of the strings "top", "bottom", "left" or "right".

1.2.26 disconnectSurfaces(surfaceNo1, side1, surfaceNo2, side2)
Removes the connection between two surface sides. Sides should be given as one of the strings "top", "bottom", "left" or "right".

1.3 Window Management
Windows are placed on surfaces, so that users can attach elements to them. Windows can be moved and all the elements they contain will be moved too.

1.3.1 newWindow(surfaceNo, x, y, width, height, name)
This call creates a new window object and returns its window number (windowNo) to be used as an identifier by future calls. The user must supply the number of the surface the window is being applied to, the x and y coordinates on the surface it should be positioned at, and its width and height.

1.3.2 newWindowWithID(ID, surfaceNo, x, y, width, height, name)
This call creates a new window object with an identifier of the user's choice and returns its window number (windowNo) to be used as an identifier by future calls. The user must supply the number of the surface the window is being applied to, the x and y coordinates on the surface it should be positioned at, and its width and height.

1.3.3 getWindowID(windowNo)
This call returns the ID of the window with the given window number.

1.3.4 setWindowID(windowNo, ID)
This call sets the ID of the window with the given window number.

1.3.5 getWindowOwner(windowNo)
This call returns the owner name of the window with the given window number.

1.3.6 getWindowAppDetails(windowNo)
This call returns the application details of the window with the given window number. This is returned as a tuple containing the application name and the instance number.

1.3.7 getWindowsByID(ID)
This call returns a list of window numbers containing all windows that have the queried ID.

1.3.8 getWindowsByOwner(owner)
This call returns a list of window numbers containing all windows that have the queried owner.

1.3.9 getWindowsByAppName(name)
This call returns a list of window numbers containing all windows that have the queried application name.

1.3.10 getWindowsByAppDetails(name, instance)
This call returns a list of window numbers containing all windows that have the queried application name and instance number.

1.3.11 moveWindow(windowNo,xDistance,yDistance)
This call moves the stated window by the given x and y distances.

1.3.12 relocateWindow(windowNo, x, y, surfaceNo)
This call relocates the stated window to the given position on the given surface.

1.3.13 getWindowPosition(windowNo)
This call returns the position of the stated window as an (x,y) tuple.

1.3.14 setWindowWidth(windowNo, width)
This call resets the width of the stated window to the chosen value.

1.3.15 setWindowHeight(windowNo, height)
This call resets the height of the stated window to the chosen value.

1.3.16 getWindowWidth(windowNo, width)
This call returns the width of the stated window.

1.3.17 getWindowHeight(windowNo, height)
This call returns the height of the stated window.

1.3.18 stretchWindowDown(windowNo, distance)
This call stretches the stated window down by the chosen number of pixels.

1.3.19 stretchWindowUp(windowNo, distance)
This call stretches the stated window up by the chosen number of pixels.

1.3.20 stretchWindowLeft(windowNo, distance)
This call stretches the stated window left by the chosen number of pixels.

1.3.21 stretchWindowRight(windowNo, distance)
This call stretches the stated window right by the chosen number of pixels.

1.3.22 setWindowName(windowNo, name)
This call sets the name of the stated window to the chosen value.

1.3.23 getWindowName(windowNo)
This call returns the name of the stated window.

1.4 Cursor Management
Cursor calls can be locked to mouse events to control a cursor on the wall. For instance, a program could track how much a mouse has moved regularly, and send a message using moveCursor() to duplicate this action on screen. When the user clicks a button on the mouse getCursorPosition() can be used to find the position of the mouse at the moment of the click and act accordingly.

1.4.1 newCursor(surfaceNo, x, y)
This call creates a new cursor object and returns its cursor number (cursorNo) to be used as an identifier by future calls. The user must supply the number of the surface the circle is being added to and the x and y coordinates of the cursor on the surface.

1.4.2 newCursorWithID(ID, surfaceNo, x, y)
This call creates a new cursor object with an identifier of the user's choice and returns its cursor number (cursorNo) to be used as an identifier by future calls. The user must supply the number of the surface the circle is being added to and the x and y coordinates of the cursor on the surface.

1.4.3 moveCursor(cursorNo, xDistance, yDistance)
This call moves the cursor with the given cursor number the requested x and y distances from its current location. Corrections are automatically made for cursor rotation.

1.4.4 testMoveCursor(cursorNo, xDistance, yDistance)
This call tests moving the cursor with the given cursor number the requested x and y distances from its current location. The hypothetical new coordinates are returned as an (x,y) tuple.

1.4.5 relocateCursor(cursorNo, x, y, surfaceNo)
This call relocates the cursor with the given cursor number to the requested x and y coordinates on the surface with the given surface number.

1.4.6 getCursorPosition(cursorNo)
This call returns the coordinates of the cursor with the given cursor number as an (x,y) tuple.

1.4.7 rotateCursorClockwise(cursorNo, degrees)
This call rotates the cursor with the given cursor number clockwise by the requested number of degrees.

1.4.8 rotateCursorAnticlockwise(cursorNo, degrees)
This call rotates the cursor with the given cursor number anticlockwise by the requested number of degrees.

1.4.9 getCursorRotation(cursorNo)
This call returns the current clockwise rotation of the cursor with the given cursor number.

1.4.10 getCursorMode(cursorNo)
This call returns the mode of the cursor with the given cursor number as a string.

1.4.11 setCursorDefaultMode(cursorNo)
This call sets the mode of the cursor with the given cursor number to "default".

1.4.12 setCursorWallMode(cursorNo)
This call sets the mode of the cursor with the given cursor number to "wall".

1.4.13 setCursorBlockMode(cursorNo)
This call sets the mode of the cursor with the given cursor number to "block".

1.4.14 setCursorScreenMode(cursorNo)
This call sets the mode of the cursor with the given cursor number to "screen".

1.4.15 showCursor(cursorNo)
This call makes the cursor with the given cursor number visible.

1.4.16 hideCursor(cursorNo)
This call makes the cursor with the given cursor number invisible.

1.4.17 isCursorVisible(cursorNo)
This call returns a boolean stating whether the cursor with the given cursor number is visible.

1.5 General Element Management
Elements are objects that can be placed on windows. There are many different types of element including rectangles, lines and text. Here are the calls that can be used for all types of element.

1.5.1 getElementID(elementNo)
This call returns the ID of the element with the given element number.

1.5.2 setElementID(elementNo, ID)
This call sets the ID of the element with the given element number.

1.5.3 getElementOwner(elementNo)
This call returns the owner name of the element with the given element number.

1.5.4 getElementAppDetails(elementNo)
This call returns the application details of the element with the given element number. This is returned as a tuple containing the application name and the instance number.

1.5.5 getElementsByID(ID)
This call returns a list of element numbers containing all elements that have the queried ID.

1.5.6 getElementsByOwner(owner)
This call returns a list of element numbers containing all elements that have the queried owner.

1.5.7 getElementsByAppName(name)
This call returns a list of element numbers containing all elements that have the queried application name.

1.5.8 getElementsByAppDetails(name, instance)
This call returns a list of element numbers containing all elements that have the queried application name and instance number.

1.5.9 getElementsOnWindow(windowNo)
This call returns a list of element numbers containing all elements that are contained within the given window.

1.5.10 getElementType(elementNo)
This call returns a string stating the type of element the given element is. (e.g. "circle")

1.5.11 showElement(elementNo)
This call makes the given element visible.

1.5.12 hideElement(elementNo)
This call makes the given element invisible.

1.5.13 checkElementVisibility(elementNo)
This call return a boolean value stating if the given element is visible.

1.5.14 removeElement(elementNo, windowNo)
This call deletes the given element from the given window.

1.6 Circle Management (Element Subclass)

1.6.1 newCircle(windowNo, x, y, radius, lineCol, fillCol, sides)
This call creates a new circle object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the circle is being added to, the x and y coordinates of the center point on the window, and its radius, line color (currently ignored), fill color and the number of sides the circle has (this defines the quality of the circle). Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.6.2 newCircleWithID(ID, windowNo, x, y, radius, lineCol, fillCol, sides)
This call creates a new circle object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the circle is being added to, the x and y coordinates of the center point on the window, and its radius, line color (currently ignored), fill color and the number of sides the circle has (this defines the quality of the circle). Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.6.3 relocateCircle(elementNo, x, y, windowNo)
This call relocates the circle with the given element number to the given position on the given window.

1.6.4 getCirclePosition(elementNo)
This call returns an (x,y) tuple containing the coordinates of the circle with the given element number.

1.6.5 setCircleLineColor(elementNo, color)
This call sets the line color of the circle with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1. THIS CALL IS CURRENTLY UNNEEDED, AS OUTLINES AREN'T DISPLAYED ON SHAPES.

1.6.6 setCircleFillColor(elementNo, color)
This call sets the fill color of the circle with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.6.7 getCircleLineColor(elementNo)
This call gets the line color of the circle with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.6.8 getCircleFillColor(elementNo)
This call gets the fill color of the circle with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.6.9 setCircleRadius(elementNo, radius)
This call sets the radius of the circle with the given element number.

1.6.10 getCircleRadius(elementNo)
This call returns the radius of the circle with the given element number.

1.6.11 setCircleSides(elementNo, sides)
This call sets the number of sides that are used to define the circle with the given element number.

1.6.12 getCircleSides(elementNo)

This call returns the number of sides that are used to define the circle with the given element number.

1.7 Line Management (Element Subclass)

1.7.1 newLine(windowNo, xStart, yStart, xEnd, yEnd, color, width)

This call creates a new line object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the line is being added to, the x and y coordinates of both the start and end points on the window, its width and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.7.2 newLineWithID(ID, windowNo, xStart, yStart, xEnd, yEnd, color, width)

This call creates a new line object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the line is being added to, the x and y coordinates of both the start and end points on the window, its width and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.7.3 getLineStart(elementNo)

This call returns the start location of the line with the given element number as a (x,y) tuple.

1.7.4 setLineStart(elementNo, x, y)

This call sets the start location of the line with the given element number.

1.7.5 getLineEnd(elementNo)

This call returns the end location of the line with the given element number as a (x,y) tuple.

1.7.6 setLineEnd(elementNo, x, y)

This call sets the end location of the line with the given element number.

1.7.7 setLineColor(elementNo, color)

This call sets the color of the line with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.7.8 getLineColor(elementNo)

This call gets the color of the line with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.7.9 setLineWidth(elementNo, width)

This call sets the width of the line with the given element number.

1.7.10 getLineWidth(elementNo)

This call gets the width of the line with the given element number.

1.8 Line Strip Management (Element Subclass)

1.8.1 newLineStrip(windowNo, x, y, color, width)

This call creates a new line strip object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the line strip is being added to, the x and y coordinates of the start point on the window, its width and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.8.2 newLineStripWithID(ID, windowNo, x, y, color, width)

This call creates a new line strip object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the line strip is being added to, the x and y coordinates of the start point on the window, its width and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.8.3 addLineStripPoint(elementNo, x, y)

This call adds a new point with the given coordinates onto the end of the line strip with the given element number.

1.8.4 addLineStripPointAt(elementNo, x, y, index)

This call adds a new point with the given coordinates at the chosen index of the line strip with the given element number.

1.8.5 getLineStripPoint(elementNo, pointNo)

This call returns the coordinates of the chosen point of the line strip with the given element number as an (x,y) tuple.

1.8.6 moveLineStripPoint(elementNo, pointNo, x, y)

This call resets the coordinates of the chosen point of the line strip with the given element number.

1.8.7 setLineStripColor(elementNo, color)

This call sets the color of the line strip with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.8.8 getLineStripColor(elementNo)

This call gets the color of the line strip with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.8.9 setLineStripWidth(elementNo, width)

This call sets the width of the line strip with the given element number.

1.8.10 getLineStripWidth(elementNo)

This call gets the width of the line strip with the given element number.

1.8.11 getLineStripPointCount(elementNo)

This call returns the number of points in the line strip with the given element number.

1.8.12 setLineStripContent(elementNo, content)

This call sets all the points of the line strip with the given element number when passed in as a list of (x,y) tuples.

1.9 Polygon Management (Element Subclass)

1.9.1 newPolygon(windowNo, x, y, lineColor, fillColor)

This call creates a new polygon object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the polygon is being added to, the x and y coordinates of the first point on the window, its line color (currently ignored) and fill color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.9.2 newPolygonWithID(ID, windowNo, x, y, lineColor, fillColor)

This call creates a new polygon object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the polygon is being added to, the x and y coordinates of the first point on the window, its line color (currently ignored) and fill color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.9.3 addPolygonPoint(elementNo, x, y)

This call adds a new point with the given coordinates onto the end of the list for the polygon with the given element number.

1.9.4 getPolygonPoint(elementNo, pointNo)

This call returns the coordinates of the chosen point in the polygon with the given element number as an (x,y) tuple.

1.9.5 movePolygonPoint(elementNo, pointNo, x, y)

This call resets the coordinates of the chosen point in the polygon with the given element number.

1.9.6 getPolygonFillColor(elementNo)

This call gets the fill color of the polygon with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.9.7 setPolygonFillColor(elementNo, color)

This call sets the fill color of the polygon with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.9.8 setPolygonLineColor(elementNo, color)

This call sets the line color of the polygon with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1. THIS CALL IS CURRENTLY UNNEEDED, AS OUTLINES AREN'T DISPLAYED ON SHAPES.

1.9.9 getPolygonLineColor(elementNo)

This call gets the line color of the polygon with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.9.10 getPolygonPointCount(elementNo)

This call returns the number of points in the polygon with the given element number.

1.10 Rectangle Management (Element Subclass)

1.10.1 newRectangle(windowNo, x, y, width, height, lineColor, fillColor)

This call creates a new rectangle object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the rectangle is being added to, the x and y coordinates of the top-left point on the window, its width, height, line color (currently ignored) and fill color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.10.2 newRectangle(ID, windowNo, x, y, width, height, lineColor, fillColor)

This call creates a new rectangle object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the rectangle is being added to, the x and y coordinates of the top-left point on the window, its width, height, line color (currently ignored) and fill color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.10.3 setRectangleTopLeft(elementNo, x, y)

This call sets the coordinates of the top-left corner of the rectangle with the given element number.

1.10.4 getRectangleTopLeft(elementNo)

This call returns the coordinates of the top-left of the rectangle with the given element number as an (x,y) tuple.

1.10.5 getRectangleTopRight(elementNo)

This call returns the coordinates of the top-right of the rectangle with the given element number as an (x,y) tuple.

1.10.6 getRectangleBottomRight(elementNo)

This call returns the coordinates of the bottom-right of the rectangle with the given element number as an (x,y) tuple.

1.10.7 getRectangleBottomLeft(elementNo)

This call returns the coordinates of the bottom-left of the rectangle with the given element number as an (x,y) tuple.

1.10.8 setRectangleWidth(elementNo, width)

This call sets the width of the rectangle with the given element number.

1.10.9 getRectangleWidth(elementNo)

This call gets the width of the rectangle with the given element number.

1.10.10 setRectangleHeight(elementNo, height)

This call sets the height of the rectangle with the given element number.

1.10.11 getRectangleHeight(elementNo)
This call gets the height of the rectangle with the given element number.

1.10.12 getRectangleFillColor(elementNo)
This call gets the fill color of the rectangle with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.10.13 setRectangleFillColor(elementNo, color)
This call sets the fill color of the rectangle with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.10.14 setRectangleLineColor(elementNo, color)
This call sets the line color of the rectangle with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1. THIS CALL IS CURRENTLY UNNEEDED, AS OUTLINES AREN'T DISPLAYED ON SHAPES.

1.10.15 getRectangleLineColor(elementNo)
This call gets the line color of the rectangle with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.11 Textured Rectangle Management (Element Subclass)

1.11.1 newTexRectangle(windowNo, x, y, width, height, texture)
This call creates a new textured rectangle object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the rectangle is being added to, the x and y coordinates of the top-left point on the window, its width, height and texture file name.

1.11.2 newTexRectangle(ID, windowNo, x, y, width, height, texture)
This call creates a new textured rectangle object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the rectangle is being added to, the x and y coordinates of the top-left point on the window, its width, height and texture file name.

1.11.3 setTexRectangleTopLeft(elementNo, x, y)
This call sets the coordinates of the top-left corner of the textured rectangle with the given element number.

1.11.4 getTexRectangleTopLeft(elementNo)
This call returns the coordinates of the top-left of the textured rectangle with the given element number as an (x,y) tuple.

1.11.5 getTexRectangleTopRight(elementNo)
This call returns the coordinates of the top-right of the textured rectangle with the given element number as an (x,y) tuple.

1.11.6 getTexRectangleBottomRight(elementNo)
This call returns the coordinates of the bottom-right of the textured rectangle with the given element number as an (x,y) tuple.

1.11.7 getTexRectangleBottomLeft(elementNo)
This call returns the coordinates of the bottom-left of the textured rectangle with the given element number as an (x,y) tuple.

1.11.8 getTexRectangleTexture(elementNo)
This call returns the name of the texture which is being used for the textured rectangle with the given element number.

1.11.9 setTexRectangleTexture(elementNo, texture)
This call sets the texture which is being used for the textured rectangle with the given element number by name.

1.11.10 setTexRectangleWidth(elementNo, width)
This call sets the width of the textured rectangle with the given element number.

1.11.11 getTexRectangleWidth(elementNo)
This call gets the width of the textured rectangle with the given element number.

1.11.12 setTexRectangleHeight(elementNo, height)
This call sets the height of the textured rectangle with the given element number.

1.11.13 getTexRectangleHeight(elementNo)
This call gets the height of the textured rectangle with the given element number.

1.12 Text Management (Element Subclass)

1.12.1 newText(windowNo, text, x, y, ptSize, font, color)
This call creates a new text block object and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the text is being added to, the string that is being displayed, the x and y coordinates of the text on the window, it's point size, font and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.12.2 newTextWithID(windowNo, text, x, y, ptSize, font, color)
This call creates a new text block object with an identifier of the user's choice and returns its element number (elementNo) to be used as an identifier by future calls. The user must supply the number of the window the text is being added to, the string that is being displayed, the x and y coordinates of the text on the window, it's point size, font and color. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.12.3 setText(elementNo, text)
This call sets the string which is shown for the text with the given element number.

1.12.4 *getText(elementNo)*
This call returns the string which is shown for the text with the given element number.

1.12.5 *setTextPosition(elementNo, x, y, windowNo)*
This call resets the position of the text with the given element number.

1.12.6 *getTextPosition(elementNo)*
This call returns the position of the text with the given element number as an (x,y) tuple.

1.12.7 *setPointSize(elementNo, pointSize)*
This call sets the point size of the text with the given element number.

1.12.8 *getPointSize(elementNo, pointSize)*
This call returns the point size of the text with the given element number.

1.12.9 *setFont(elementNo, font)*
This call sets the font being used for the text with the given element number. This currently has to be one of the fonts that the server supports. ("Free Serif", "Free Mono" or "Free Sans")

1.12.10 *getFont(elementNo)*
This call returns the font being used for the text with the given element number.

1.12.11 *setTextColor(elementNo, color)*
This call sets the color of the text with the given element number. Colors should be passed in as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.12.12 *getTextColor(elementNo)*
This call returns the color of the text with the given element number. Colors are returned as an (r,g,b,α) tuple where the values of r, g, b and α are between 0 and 1.

1.13 General Calls
1.13.1 *quit()*
This call stops the looping of the message sender in preparation to close the program and sends the server a message to close as well.

1.13.2 *quitClientOnly()*
This call stops the looping of the message sender in preparation to close the program.

1.13.3 *uploadImage(file)*
This call sends the image file that the user stated the location of to the server so that it can be used within future programs as a texture.

1.13.4 *getSavedImages()*
Returns a list of all the images stored on the server side.

1.13.5 *deleteImage(filename)*
Deletes the image with the given file name from the server.

2. HELLO WORLD
Here is a hello world program that can be run after the walls have been set up in the configuration program. It assumes the user has set up surface 1 and displays the text "Hello World" in the center.

```
from messageSender import *

#Create Connection
sender = messageSender()

#Log into the server
sender.login("username")
sender.setapp("helloworld")

#Check the width and height of the surface
height = sender.getSurfacePixelHeight(1)
width = sender.getSurfacePixelWidth(1)

#Create a window that covers the whole
#surface
win = sender.newWindow(1, 0, height, width,
        height, "HWwin")

#Render the words "Hello World" in the
#middle of the window.
sender.newText(win, "Hello World",
        height/2-150, width/2-25, 60,
        "Free Sans", (1,1,1,1))
```

3. REST API
If the server is run using "restServer.py" instead of "server.py" it will take take http messages instead of standard API calls. These are formatted as "/api/<CALL NAME>". Ones which add or change information are "POST" requests while those which just request information are "GET" requests. Parameters should be passed into these REST calls as JSON structures. Returned information is returned as a JSON structure.
Logging in takes place automatically when using the REST server, so there is no need to call login() or setApp()

4. SENDING IMAGES WITH REST API
To display images to the rest API, a different method is used than the usual one that exists outside REST. Images are sent in the same calls that ask for them to be displayed. So, instead of a reference to the texture being included in the JSON for a call to /api/newTexRectangle, /api/newTexRectangleWithID, or /api/setTexRectangleTexture, the calls take the image data itself (converted to base64) and the file extension. These JSON parameters should be called "textureData" and "extension" respectively. I have created a python library called "jsonImageSender.py" which contains the class jsonImageSender which allows for easy sending of images. Here are the calls that this class provides.

4.0.6 *Constructor - jsonImageSender(host,port)*

Creates an instance of the class and tells it the host name and port number for the rest server for use in the provided call functions.

4.0.7 *newTexRectangle(windowNo, filename, x, y, width, height)*

Creates a base64 representation of the image with the chosen filename and sends it to the rest server to be displayed with the stated parameters.

4.0.8 *newTexRectangleWithID(ID,windowNo, filename, x, y, width, height)*

Creates a base64 representation of the image with the chosen filename and sends it to the rest server to be displayed with the stated parameters.

4.0.9 *setTexRectangleTexture(elementNo, filename)*

Creates a base64 representation of the image with the chosen filename and sends it to the rest server to replace the texture being used for the existing texRectangle with the stated element number.

5. REST HELLO WORLD

This example would be run in a UNIX terminal using "curl" assuming that the server is running on the same computer.

```
curl -i -H "Content-Type: application/json"
        -X POST -d "{\"surfaceNo\":0,
        \"x\":0,\"y\":512,\"width\":512,
        \"height\":512,\"name\":\"HWwin\"}"
        http://localhost:5000/api/newWindow
curl -i -H "Content-Type: application/json"
        -X POST -d "{\"windowNo\":1,
        \"text\":\"Hello World\",\"x\":300,
        \"y\":300,\"pt\":60,
        \"font\":\"Free Sans\",
        \"color\":\"1:1:1:1\"}"
        http://localhost:5000/api/newText
```