CMX

Using update

API

<https://cmxlocationsandbox.cisco.com/apidocs/location-api#Clients-History-API-GET-This-API-returns-history-of-client-by-macaddress>

**[Clients History API](https://cmxlocationsandbox.cisco.com/apidocs/location-api" \l "Clients-History-API)**

* [**List Methods**](https://cmxlocationsandbox.cisco.com/apidocs/location-api)
* [**Expand Methods**](https://cmxlocationsandbox.cisco.com/apidocs/location-api)
* GET This API returns unique macaddresses seen on a given day on a floor or zone./api/location/v1/history/uniqueclientsbyhierarchy
* GET This API returns compact history of client by date and ipv4address/api/location/v1/historylite/byipaddress/:ipv4address
* GET This API returns compact history of client by date and username/api/location/v1/historylite/byusername/:username
* GET This API returns history of client by macaddress/api/location/v1/history/clients/:macaddress
* GET This API returns compact history of client by date and macaddress/api/location/v1/historylite/clients/:macaddress
* GET This API returns history for all clients./api/location/v1/history/clients

Matt Fowlers Original Code

# Get the image file of the floor the client is on from CMX, save it to a file, then load it in StingIO so we can work with it in memory.

# We save it to a file first so that next time we don't have to pull an image from CMX. Speed things up and reduces load on CMX.

image = cmxContent(cmxAddr+urlFloorImage + client["mapInfo"]["image"]["imageName"])

file = image\_path + client["mapInfo"]["image"]["imageName"]

fh = open(file, "w+")

fh.write(image)

fh.close()

fh = open(file, "rb")

image = storeMemory(fh.read()).encode("base64").strip()

# Split the Campus>Building>Floor string into it's components.

# This is useful if you want to print out the location, particularly the floor as floor plans may be similar and hard to distinguish via the image alone.

hierarchy = client["mapInfo"]["mapHierarchyString"].split('>')

# Now, plot the user's location on the image.

# Read the floor image from memory with pyplot. Pyplot uses PIL to support jpeg.

im = plt.imread(StringIO(image.decode('base64')), format='jpeg')

# Some images were showing red like http://stackoverflow.com/questions/21641822/ and this seems to fix it.

convertedim=Image.open(StringIO(image.decode('base64'))).convert('P')

# Draw a plot over the image that is the same size as the image.

implot = plt.imshow(convertedim, extent=[0, client["mapInfo"]["floorDimension"]["width"], 0, client["mapInfo"]["floorDimension"]["length"]], origin='lower', aspect=1)

# Mark the client's coordinates that we received from CMX.

# The first line will draw a dot at the x,y location and the second and third lines will draw circles around it.

plt.scatter([str(client["mapCoordinate"]["x"])], [str(client["mapCoordinate"]["y"])], facecolor='r', edgecolor='r')

plt.scatter([str(client["mapCoordinate"]["x"])], [str(client["mapCoordinate"]["y"])], s=1000, facecolors='none', edgecolor='r')

plt.scatter([str(client["mapCoordinate"]["x"])], [str(client["mapCoordinate"]["y"])], s=2000, facecolors='none', edgecolor='r')

plt.scatter([str(client["mapCoordinate"]["x"])], [str(client["mapCoordinate"]["y"])], s=3500, facecolors='none', edgecolor='r')

# Currently the plot is the same size as the image, but the scale is off so we need to correct that.

ax = plt.gca()

ax.set\_ylim([0,client["mapInfo"]["floorDimension"]["length"]])

ax.set\_xlim([0,client["mapInfo"]["floorDimension"]["width"]])

# The plot starts 0,0 from the bottom left corner but CMX uses the top left.

# So, we need to invert the y-axis and, to make it easier to read, move the x axis markings to the top (if you choose to show them).

ax.set\_ylim(ax.get\_ylim()[::-1])

ax.xaxis.tick\_top()

# Use this to decide whether you want to show or hide the axis markings.

plt.axis('off')

# Save our new image with the plot overlayed to memory. The dpi option here makes the image larger.

plt.savefig(buff, format='png', dpi=500)

# Get the new image.

newimage = buff.getvalue().encode("base64").strip()