```python
import hou
import os

from big_framework import string_processor

import ui_attempt
import lod_and_bake

# so houdini doesn't use the precompiled:
reload(ui_attempt)
reload(lod_and_bake)

# so to copy and paste all the code means I have to delete 'ui_attempt.' preceding etc.



def os_path_join_fix(*args): # in the version of Python that Houdini has, os_path_join_fix is broken
    a_path = ""
    if len(args) == 0:
        return a_path
    else:
        slash = os.path.sep
        for item in args:
            a_path += item + slash

        return a_path[:-1] # so there isn't a final slash in the end


def get_file_scan(a_path): # versus 'get_maps' which is risky since it requires knowing all the possible map names (instead, get all files, and take maps you want that exist with get_maps_of_name_type_and_res
    file_scan_list = list()
    scan_list = os.listdir(a_path)
    for material_name in scan_list:
        a_path = os_path_join_fix(a_path, material_name)
        if os.path.isdir(a_path) == False:
            file_scan_list.append(material_name)
    return file_scan_list

def get_child_from_parent_node(parent_node_path, child_name): # exception handling expected by caller
    child_node_path = "{}/{}".format(parent_node_path, child_name)
    child_node = hou.node(child_node_path)
    return child_node


def get_file_extension(file_path_or_name):
    return os.path.splitext(file_path_or_name)[1]

def get_megascans_resolution_str_from_resolution(resolution): # e.g. given 4 * 1024, return "4K"
    return str(resolution * 1024) + "K"

def get_resolution_from_megascans_resolution_str(megascans_resolution_str): # given e.g. "4K", return 4 * 1024. Good to have a function because this logic could change in the future
    return int(megascans_resolution_str[:-1]) * 1024


def get_megascans_resolution(file_path_or_name, return_int = False):
    first_underscore_index = file_name.find("_")
    second_underscore_index = first_underscore_index + 1 + file_name[first_underscore_index + 1:].find("_")
    megascans_resolution_str = file_name[first_underscore_index + 1: second_underscore_index] # e.g. "4K"

    if return_int == False:
        return megascans_resolution_str
    else:
        return get_resolution_from_megascans_resolution_str(megascans_resolution_str) # should I remove this functionality and leave it to the user?


def get_highest_resolution(megascans_folder_scan): # i.e. ONLY maps don't have to be passed
    resolution_list = list()

    for file_name in megascans_folder_scan:
        try: # may or may not be map
            megscans_resolution_int = get_megascans_resolution(file_name, True) # this gives e.g. "4K"
            resolution_list.append(megascans_resolution_int)
        except: # if not map (i.e. no resolution)
            pass

    if len(resolution_list) == 0:
        return None # I think that's the cleanest thing to do, as oppose than returning a default resolution (let that be decided elsewhere)
    else:
        return max(a_list) * 1024



def get_maps_of_name_type_and_res(file_scan, desired_map_name, file_extension_list = None, resolution_list = None): # in descending order
    existing_maps = [file_name for file_name in file_scan if desired_map_name in file_name]

    sorted_maps = existing_maps # for clarity
    if file_extension_list != None: # first sort
        sorted_maps = sorted(sorted_maps, key = lambda map_name: file_extension_list.index(get_file_extension(map_name)))

    if resolution_list != None: # second sort
        sorted_maps = sorted(sorted_maps, key = lambda map_name: resolution_list.index(get_megascans_resolution(map_name, False)))

    return sorted_maps




def get_megascans_asset_name(megascans_folder_path):
    megascans_folder_name = os.path.basename(megascans_folder_path) # just in case
    megascans_asset_name = megascans_folder_name[megascans_folder_name.rfind("_") + 1:] # i.e. given rock_assembly_S01ez, returns S01ez
    return megascans_asset_name


def get_node_with_throw_error(node_path): # made to stop repeated code
    a_node = hou.node(node_path)
    if a_node == None:
        node_name = node_path[node_path.rfind("/") + 1:]
        raise Exception("{} not found at {}".format(node_name, node_path))
    return a_node


def get_nodes(megascans_asset_fix_subnet_node): # assumes using certain version of Bridge
    megascans_asset_subnet_path = megascans_asset_fix_subnet_node.path()
    asset_geometry_path = "{}/Asset_Geometry".format(megascans_asset_subnet_path)
    asset_geometry_node = hou.node(asset_geometry_path)
    asset_material_path = "{}/Asset_Material".format(megascans_asset_subnet_path)
    asset_material_node = hou.node(asset_material_path)

    # good to check that a megascans subnet is even selected before doing the rest
    if asset_geometry_node == None or asset_geometry_node == None:
        raise Exception("'Asset_Geometery' or 'Asset_Material' aren't children of {}.\nAre you sure you've selected a Megascans Asset Subnetwork?".format(megascans_asset_subnet_path))

    file_node_path = "{}/Asset_Geometry/file1".format(megascans_asset_subnet_path) # more adaptable to give path, instead of getting as child from Asset_Material
    transform_node_path = "{}/Asset_Geometry/transform1".format(megascans_asset_subnet_path) # ditto ^
    file_node = get_node_with_throw_error(file_node_path)
    transform_node = get_node_with_throw_error(transform_node_path)


    # Is it worth it to get these here? Step 1 and 2 can carry on without these (also transform not necessary in the baove)
    rs_material_builder_node = asset_material_node.children()[0]
    if rs_material_builder_node.type().name() != "redshift_vopnet":
        raise Exception("Expected node in Asset Geometry to be of type 'redshift_vopnet'")

    redshift_material_node = None
    for child in rs_material_builder_node.children():
        if child.type().name() == "redshift_material":
            redshift_material_node = child
            break
    if redshift_material_node == None:
        raise Exception("Cannot find node of type 'redshift_material' in RS Material Builder")

    return asset_geometry_node, asset_material_node, file_node, transform_node, rs_material_builder_node, redshift_material_node

```

```python

def replace_substring_with_count(a_string, substring_to_replace, count):
    while substring_to_replace in a_string:
        a_string = a_string.replace(substring_to_replace, str(count), 1)
        count += 1
    return a_string, count

def add_to_megascans_material_node_setup(rs_material_builder_node, map_name_and_node_setup_dict, map_name_and_export_paths_dict, current_bump_blender_layer):
    for map_name in map_name_and_export_paths_dict.keys(): # have to get keys again since htey've changed
        try:
            node_setup_string = map_name_and_node_setup_dict[map_name]
        except KeyError: # only error it could be
            raise Exception("map_name_and_node_setup_dict does not contain the node setup for map_name: {}".format(map_name))
        else:
            a_export_path = map_name_and_export_paths_dict[map_name]
            node_setup_string, current_bump_blender_layer = replace_substring_with_count(node_setup_string, "{bump_blender_layer}", current_bump_blender_layer)
            node_setup_string = node_setup_string.format(export_path = a_export_path.replace(" ", "%20")) # using format instead of replace, just for the sake of that's how I would've done the above
            string_processor(rs_material_builder_node, node_setup_string)




class MegascansAsset: # this seems clean. Makes sense to make a class to hold all this information while interacting with the GUI (rather than pass it around or use global variables)
    def __init__(self, megascans_asset_subnet):
        self.megascans_asset_subnet = megascans_asset_subnet

        # Gets all necessary nodes (TODO: identify exactly what nodes aren't retrieved here), the goal is that this also identifies if there's anything that'll stop Step 1, 2 and 3 from running (i.e. a Megascans Asset that has been modified)
        self.asset_geometry_node, self.asset_material_node, self.file_node, self.transform_node, self.rs_material_builder_node, self.redshift_material_node = get_nodes(self.megascans_asset_subnet) # remember in tuple unpacking, any name can be used i.e. i've added on self

        self.megascans_asset_folder_path = os.path.dirname(self.file_node.parm("file").eval())
        self.megascans_asset_name = get_megascans_asset_name(self.megascans_asset_folder_path)

        self.file_scan = get_file_scan(self.megascans_asset_folder_path)

        # Executing of the above with no errors means it's confirmed it's a megascans asset, and should be time to call the UI. Perhaps edit the above error code to throw a hou.ui.displayMessage if anything goes wrong (rather than the existing exceptions) - maybe pull this off with a try except?

    def execute_fix(self, polyreduce_percentage_float, maps_to_bake_dict, bake_resolution_str, use_temp_displacement_bool): # can't think of a better name
        # Step 1 and 2 are housed in this subnet node
        fix_subnet_node = self.megascans_asset_subnet.createNode("subnet", "Megascans_Fixer_Subnet") # Feel free to change name

        #---------------------------------------------
        # Step 1) Make Custom LOD
        #print("Step 1 begins")

        customlod_name = self.megascans_asset_name + "_LOD_custom_{}percent.fbx".format(polyreduce_percentage_float)
        customlod_path = os_path_join_fix(self.megascans_asset_folder_path, customlod_name)

        highpoly_name = get_maps_of_name_type_and_res(self.file_scan, "High", file_extension_list = [".fbx"])[0] # pick best from sorted, which is at index 0
        highpoly_path = os_path_join_fix(self.megascans_asset_folder_path, highpoly_name)

        a_lod_object = lod_and_bake.LOD(highpoly_path, polyreduce_percentage_float, customlod_path)
        a_lod_object.create_and_execute_in_houdini(fix_subnet_node)

        #---------------------------------------------
        # Step 2) Bake Custom Maps, and give dictionary with their map names and export paths
        #print("Step 2 begins")

        # for clarity
        bake_resolution_x_and_y = get_resolution_from_megascans_resolution_str(bake_resolution_str)

        #maps_to_bake_dict = lod_and_bake.Bake.maps_to_bake_dict_template
        #maps_to_bake_dict["Displacement"] = True
        #maps_to_bake_dict["Vector Displacement"] = True
        #The above has been commented out because it is now being passed in by the UI

        export_name_prefix = self.megascans_asset_name + "_" + bake_resolution_str
        a_bake_object = lod_and_bake.Bake(highpoly_path, customlod_path, maps_to_bake_dict, bake_resolution_x_and_y, bake_resolution_x_and_y, self.megascans_asset_folder_path, export_name_prefix = export_name_prefix)
        map_name_and_export_paths_dict = a_bake_object.create_and_execute_in_houdini(fix_subnet_node)

        #-----------------------------------------------
        # Step 3) Configure and Modify Megascans Material's Node Setup (enable tessalation, displacement etc. and edit node setup)
        #print("Step 3 begins")



        # Enable Tessellation, Displacement, and set Displacement Scale
        self.asset_geometry_node.parm("RS_objprop_rstess_enable").set(1)
        self.asset_geometry_node.parm("RS_objprop_displace_enable").set(1)
        displacement_scale = self.transform_node.parm("scale").eval() # retrieved from transform_node after file import
        self.asset_geometry_node.parm("RS_objprop_displace_scale").set(displacement_scale)


        # Create Bump Blender (note, I have not changed layer blend weights like I did last time!) in Megascans Material's Node Setup
        string_processor(self.rs_material_builder_node, "cBumpBlender-bump_blender i0 e{} i2".format(self.redshift_material_node.name()))
        current_bump_blender_layer = 0 # assuming 'Base' on BumpBlender doesn't need to be used


        # Hardcoded logic on Megascans Material's Node Setup
        map_name_and_export_paths_dict_keys = map_name_and_export_paths_dict.keys() # so I don't have to get the keys again (probably not worth it)
        if "Vector Displacement" in map_name_and_export_paths_dict_keys and "Displacement" in map_name_and_export_paths_dict_keys: # if both there, only set up Vector Displacement
            map_name_and_export_paths_dict.pop("Displacement")

        if "Normal" in map_name_and_export_paths_dict_keys:
            for child in self.rs_material_builder_node.children(): # destroy the legacy normal map
                if child.type().name() == "redshift::NormalMap":
                    child.destroy()
                    break


        # Add to Megascans Material's Node Setup
        # Configure Map Name and Node Setup Dict
        map_name_and_node_setup_dict = dict()
        map_name_and_node_setup_dict["Displacement"] = "@edisplacement!tex0:{export_path} @eDisplacement1!map_encoding:1"
        map_name_and_node_setup_dict["Vector Displacement"] = "@edisplacement!tex0:{export_path} @eDisplacement1!map_encoding:0"
        #map_name_and_node_setup_dict["Bump Map"] = "cTextureSampler-bump!tex0:{export_path}!color_multiplierr:0.2!color_multiplierg:0.2!color_multiplierb:0.2 i0 cBumpMap-bump_for_bump i0 ebump_for_bump i0 ebump_blender nbaseInput{bump_blender_layer}"
        #map_name_and_node_setup_dict["Normal"] = "cNormalMap-normal!tex0:{export_path} i0 cBumpMap-bump_for_normal!inputType:1 i0 ebump_for_normal i0 ebump_blender nbumpInput{bump_blender_layer}"

        add_to_megascans_material_node_setup(self.rs_material_builder_node, map_name_and_node_setup_dict, map_name_and_export_paths_dict, current_bump_blender_layer)


        #-----------------------------------------------
        # Final touches

        self.file_node.parm("file").set(customlod_path)

        # Layout the subnet that holds everything, and set display flag to off
        fix_subnet_node.layoutChildren()
        fix_subnet_node.setDisplayFlag(False)

        # Layout the thing that holds the subnet
        self.megascans_asset_subnet.layoutChildren()

        # Set Network Editor pane to be where you started (at the location of the megascans asset node - as oppose to inside the fix_subnet_node)
        network_editor = [pane for pane in hou.ui.paneTabs() if isinstance(pane,
hou.NetworkEditor) and pane.isCurrentTab()][0] # assuming just one.
        # ^ as per: https://forums.odforce.net/topic/12406-getting-the-current-active-network-editor-pane/, doesn't seem like there's a better way to do it nowadays
        network_editor.setCurrentNode(self.megascans_asset_subnet)

        hou.ui.displayMessage("Done successfully!") # feel free to change



def main():
    selected_node_list = hou.selectedNodes()
    if len(selected_node_list) != 1:
        raise Exception("Zero or Multiple nodes selected. Are you sure you've selected a single Megascans Asset Subnetwork?")

    selected_node = selected_node_list[0] # to access later on
    megascans_asset_subnet = selected_node # assumming

    try:
        megascans_asset_object = MegascansAsset(megascans_asset_subnet)
    except Exception as exception:
        hou.ui.displayMessage("Error Occured:\n\n{}\n\nPlease try again".format(exception))
        raise SystemExit # good practice way to exit according to https://stackoverflow.com/questions/19747371/python-exit-commands-why-so-many-and-when-should-each-be-used

    ui = ui_attempt.MegascansFixerDialog(megascans_asset_object)
    ui.show()

    # the above handles calling the 'execute_fix' method upon the 'Go!' button being pressed






#main()

```