Nanome-Plugin-API Documentation

Nanome

May 19, 2021

Contents

The Nanome Plugin System is a Python-based API that allows users to connect 3rd party tools into the Nanome Virtual Reality Software Tool for Collaborative Molecular Modeling.

CHAPTER 1

Table of Contents

1.1 Architecture

The overall architecture of this plugin system is designed to enable plugin creators to iterate fast and efficiently when developing, improving, or debugging a plugin for Nanome.

If you have any feedback or question, don't hesitate to contact us or to directly contribute to our Github

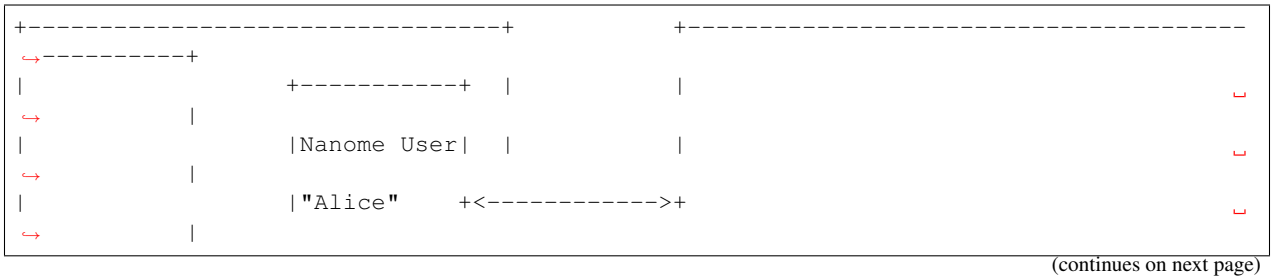
1.1.1 Development iterations

As a result of this flexible architecture, no need to restart Nanome if your plugin crashes, or if you need to improve it:

1. Stop your plugin. All sessions connected to it will be disconnected.
2. Modify the python script
3. Restart it
4. Reconnect to it in Nanome. Using the 2D mode of Nanome might be useful in order to reconnect and test faster without having to wear your VR headset everytime.

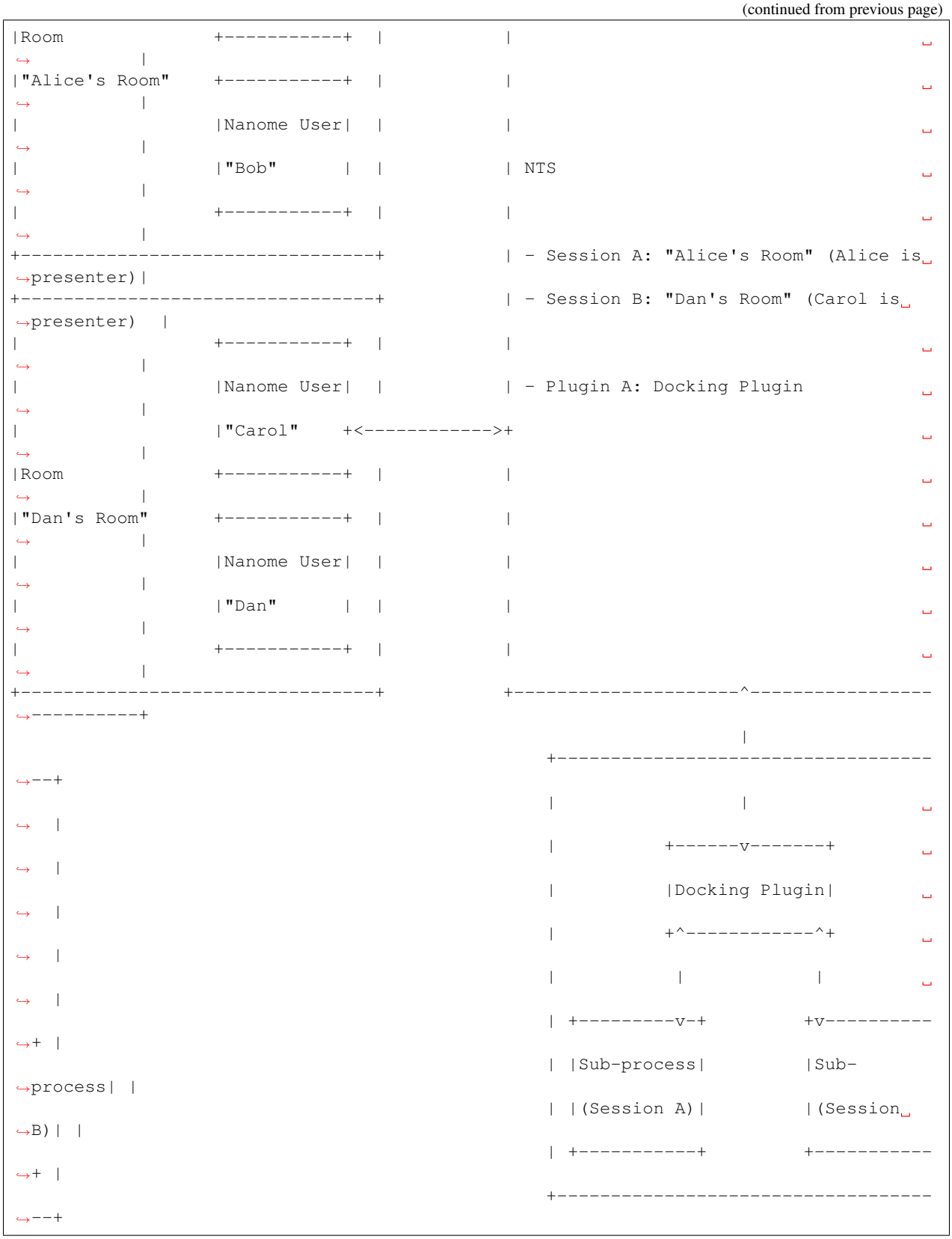
1.1.2 How it works

Here is a simple way to represent the Plugin System architecture:



(continues on next page)

(continued from previous page)



1. NTS (the plugin server) is aware of which plugins and sessions are connected to it, and who is the presenter of

each session.

2. A session asks to connect to a plugin
3. NTS transfers the request to the target plugin
4. The plugin creates a subprocess on its computer, and instantiates its plugin class
5. The subprocess replies to its main process, which transfers the reply to NTS, which transfers the reply to the room presenter
6. Connection is established until the presenter requests a disconnection or the plugin is stopped.

NB: A plugin cannot talk to a Nanome session/user before being connected to it. NB2: Communications are encrypted from Nanome to NTS and from NTS to Plugins

1.2 Basics

1.2.1 Plugin description

The parameters of nanome.Plugin define how your plugin will appear in the list:

```
plugin = nanome.Plugin(name, description, category, has_advanced)
```

- *category* will define in which category the plugin will be when Nanome User clicks on the plugin filter drop-down. This is currently unsupported.
- *has_advanced* defines if an "Advanced Settings" button should be displayed when user selects the plugin in Nanome

Or, if using the one-line plugin setup:

```
nanome.Plugin.setup(name, description, category, has_advanced, plugin_class, host,␣
↪port, key_file)
```

1.2.2 Entry points

Overriding these functions in your plugin will give you several entry points:

```
def start(self):
    """
    | Called when user "Activates" the plugin
    """
    pass

def update(self):
    """
    | Called when when instance updates (multiple times per second)
    """
    pass

def on_run(self):
    """
    | Called when user presses "Run"
    """
    pass
```

(continues on next page)

(continued from previous page)

```
def on_stop(self):
    """
    | Called when user disconnects or plugin crashes
    """
    pass

def on_advanced_settings(self):
    """
    | Called when user presses "Advanced Settings"
    """
    pass

def on_complex_added(self):
    """
    | Called whenever a complex is added to the workspace.
    """
    pass

def on_complex_removed(self):
    """
    | Called whenever a complex is removed from the workspace.
    """
    pass

def on_presenter_change(self):
    """
    | Called when room's presenter changes.
    """
    pass

def on_advanced_settings(self):
    """
    | Called when user presses "Advanced Settings"
    """
    pass

def on_complex_added(self):
    """
    | Called whenever a complex is added to the workspace.
    """
    pass

def on_complex_removed(self):
    """
    | Called whenever a complex is removed from the workspace.
    """
    pass

def on_presenter_change(self):
    """
    | Called when room's presenter changes.
    """
    pass
```

1.3 Workspace API

1.3.1 Request entire workspace in deep mode

```
def on_run(self):
    self.request_workspace(self.on_workspace_received)

def on_workspace_received(self, workspace):
    pass
```

1.3.2 Request a list of specific complexes in deep mode

```
def on_run(self):
    self.request_complexes([1, 6, 5], self.on_complexes_received) # Requests␣
↪complexes with ID 1, 6 and 5

def on_complexes_received(self, complex_list):
    pass
```

1.3.3 Request all complexes in the workspace in shallow mode

```
def on_run(self):
    self.request_complex_list(self.on_complex_list_received)

def on_complex_list_received(self, complex_list):
    pass
```

1.3.4 Update workspace to match exactly

```
def on_workspace_received(self, workspace):
    # ...
    # Do something with workspace
    # ...
    self.update_workspace(workspace)
```

1.3.5 Add to workspace

```
def on_run(self):
    # ...
    # Create new complexes
    # ...
    self.add_to_workspace([new_complex1, new_complex2])
```

1.3.6 Update specific structures

In shallow mode:

```
def on_complex_list_received(self, complex_list):
    # ...
    # Do something with shallow structures, i.e. move them, rename them
    # ...
    self.update_structures_shallow([complex, atom, residue])
```

In deep mode:

```
def on_workspace_received(self, complex_list):
    # ...
    # Do something with deep structures, i.e. move them, rename them
    # ...
    self.update_structures_deep([complex])
```

1.3.7 Structure

Deep / Shallow

Nanome has two molecular structure transmission mode: Deep and Shallow. Their goal is to make data transmission faster by requesting only the data needed.

- **Deep mode** will request/send the structure and its entire content. E.g. a molecule in deep mode will contain its name and any other property it might have + all its chains, residues, atoms and bonds
- **Shallow mode** will request/send only the structure itself. E.g. a molecule in shallow mode will only contain its name and any other property it might have

Whether a command requests one mode or the other is described in this documentation.

Structure hierarchy

Molecular structures are organized like so:

- **Workspace**
- —-**Complex**
- ——— **Molecule**
- ————— **Chain**
- ——————— **Residue**
- ————————— **Atom**
- ————————— **Bond**

A complex is a group of molecules and has a position and rotation. In Nanome, the user can switch between the molecules of a complex using the frame slider, in the information menu.

Index

Each molecular structure has an index available as a base property.

This index is a unique identifier for structures uploaded to Nanome. However, if a structure hasn't been added to Nanome's workspace yet, its index will be -1

To access this index:

```
if my_structure.index == -1:
    Logs.message("This structure hasn't been uploaded to Nanome")
```

1.4 User Interface API

1.4.1 Plugin Menu Creator

In order to make menu creation easier, we provide a tool called StackStudio.

Todo: Write how to integrate plugin menu creator menus in a plugin

1.4.2 API

The UI API can be used to create a Menu from scratch or to interact with any menu or UI element generator by the Plugin Menu Creator.

UI elements are organized like so:

- **Menu** - Contains its size, title, enabled state, etc.
- —- **Root** - Main Layout Node
- ———- **Layout** Node - Contains positioning information, orientation, etc.
- ——————- **Content** - Button/Slider/Text Input/etc.
- ——————- **Children** Layout Nodes - A layout node can contain other Layout Nodes
- ————————- etc.

A menu hierarchy is created by placing *LayoutNode* under each other, and changing their orientations and sizes.

Currently available UI elements are:

- *Button*
- *Slider*
- *Label*
- *TextInput*
- *Image*
- *Mesh*
- *UIList*
- *Dropdown*

Following is an example of manual UI creation:

```
import nanome
from nanome.util import Logs
import sys
import time

# Config
```

(continues on next page)

(continued from previous page)

```
NAME = "UI Plugin"
DESCRIPTION = "A simple plugin demonstrating how plugin system can be used to extend␣
↪Nanome capabilities"
CATEGORY = "File Import"
HAS_ADVANCED_OPTIONS = False

# Plugin

def menu_closed_callback(menu):
    Logs.message("Menu closed: " + menu.title + " " + str(menu.enabled))

def menu_opened_callback(menu):
    Logs.message("Menu opened: " + menu.title + " " + str(menu.enabled))

def slider_changed_callback(slider):
    Logs.message("slider changed: " + str(slider.current_value))

def dropdown_callback(dropdown, item):
    Logs.message("dropdown item selected: " + str(item.name))

def slider_released_callback(slider):
    Logs.message("slider released: " + str(slider.current_value))

def text_changed_callback(textInput):
    Logs.message("text input changed: " + str(textInput.input_text))

def text_submitted_callback(textInput):
    Logs.message("text input submitted: " + str(textInput.input_text))

class UIPlugin(nanome.PluginInstance):
    def create_callbacks(self):
        def spawn_menu_callback(button):
            Logs.message("button pressed: " + button.text.value.idle)
            self.update_content(button)
            self.spawn_sub_menu()

        self.spawn_menu_callback = spawn_menu_callback

        def hover_callback(button, hovered):
            Logs.message("button hover: " + button.text.value.idle, hovered)

        self.hover_callback = hover_callback

        def select_button_callback(button):
            button.selected = not button.selected
            Logs.message("Prefab button pressed: " + button.text.value.idle + " " +␣
↪str(button._content_id))
            self.update_content(button)

        self.select_button_callback = select_button_callback

        def loading_bar_callback(button):
            Logs.message("button pressed: " + button.text.value.idle)

            self.loadingBar.percentage += .1
            self.loadingBar.title = "TITLE"
```

(continues on next page)

(continued from previous page)

```
            self.loadingBar.description = "DESCRIPTION " + str(self.loadingBar.
↪percentage)

            self.update_content(self.loadingBar)

        self.loading_bar_callback = loading_bar_callback

    def start(self):
        self.integration.import_file = self.import_file
        Logs.message("Start UI Plugin")
        self.create_callbacks()

    def import_file(self, request):
        self.on_run()

    def on_run(self):
        Logs.message("Run UI Plugin")
        menu = self.rebuild_menu()
        self.update_menu(menu)

    def rebuild_menu(self):
        self.menu = nanome.ui.Menu()
        menu = self.menu
        menu.title = "Example UI Plugin"
        menu.width = 1.0
        menu.height =  1.0
        menu.register_closed_callback(menu_closed_callback)
        self.tab1 = self.create_tab1()
        self.tab2 = self.create_tab2()
        self.tab2.enabled = False
        self.tab_buttons = self.create_tab_buttons()
        menu.root.add_child(self.tab_buttons)
        self.tabs = menu.root.create_child_node()
        self.tabs.add_child(self.tab1)
        self.tabs.add_child(self.tab2)
        return menu

    def spawn_sub_menu(self):
        menu = nanome.api.ui.Menu(self.menu_index, "Menu " + str(self.menu_index))
        menu.register_closed_callback(menu_closed_callback)
        menu.width = 0.5
        menu.height = 0.5
        if self.previous_menu != None:
            ln = self.previous_menu.root.create_child_node()
            ln.add_new_label(str(self.menu_index - 1))
            self.update_menu(self.previous_menu)

        root = menu.root
        button_node = root.create_child_node("button_node")
        button = button_node.add_new_button("button")
        button.register_pressed_callback(self.select_button_callback)

        self.update_menu(menu)
        self.menu_index += 1
        self.previous_menu = menu

    def create_tab1(self):
```

(continues on next page)

(continued from previous page)

```
        self.menu_index = 1
        self.previous_menu = None

        content = nanome.ui.LayoutNode()
        ln_contentBase = nanome.ui.LayoutNode()
        ln_label = nanome.ui.LayoutNode()
        ln_button = nanome.ui.LayoutNode()
        ln_slider = nanome.ui.LayoutNode()
        ln_textInput = nanome.ui.LayoutNode()
        ln_list = nanome.ui.LayoutNode()

        content.forward_dist = .02
        content.layer = 1

        ln_label.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_label.padding = (0.01, 0.01, 0.01, 0.01)
        ln_label.forward_dist = .001

        label = nanome.ui.Label()
        label.text_value = "Press the button..."
        label.text_color = nanome.util.Color.White()

        Logs.message("Added Label")

        ln_button.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_button.padding = (0.01, 0.01, 0.01, 0.01)
        ln_button.forward_dist = .001

        #super styled button
        button = nanome.ui.Button()
        button.name = "OpenSubMenu"
        b_t = button.text
        b_t.active = True
        b_t.value.set_all("Spawn menu")
        b_t.auto_size = False
        b_t.size = .6
        b_t.underlined = True
        b_t.ellipsis = True
        b_t.color.idle = nanome.util.Color.Red()
        b_t.color.highlighted = nanome.util.Color.Blue()
        b_t.bold.set_all(False)
        b_t.padding_left = .5
        b_t.vertical_align = nanome.util.enums.VertAlignOptions.Middle
        b_t.horizontal_align = nanome.util.enums.HorizAlignOptions.Left
        b_m = button.mesh
        b_m.active = True
        b_m.color.idle = nanome.util.Color.Blue()
        b_m.color.highlighted = nanome.util.Color.Red()
        b_o = button.outline
        b_o.active = True
        b_o.color.idle = nanome.util.Color.Red()
        b_o.color.highlighted = nanome.util.Color.Blue()
        b_t = button.tooltip
        b_t.title = "spawn a submenu"
        b_t.content = "it is useless"
        b_t.positioning_target = nanome.util.enums.ToolTipPositioning.center
        button.register_pressed_callback(self.spawn_menu_callback)
```

(continues on next page)

(continued from previous page)

```
        button.register_hover_callback(self.hover_callback)

        Logs.message("Added button")

        ln_slider.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_slider.padding = (0.01, 0.01, 0.01, 0.01)
        ln_slider.forward_dist = .001

        slider = nanome.ui.Slider()
        slider.register_changed_callback(slider_changed_callback)
        slider.register_released_callback(slider_released_callback)

        Logs.message("Added slider")

        ln_textInput.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_textInput.padding = (0.01, 0.01, 0.01, 0.01)
        ln_textInput.forward_dist = .001

        textInput = nanome.ui.TextInput()
        textInput.max_length = 30
        textInput.register_changed_callback(text_changed_callback)
        textInput.register_submitted_callback(text_submitted_callback)
        textInput.number = True
        textInput.text_color = nanome.util.Color.Blue()
        textInput.placeholder_text_color = nanome.util.Color.Red()
        textInput.background_color = nanome.util.Color.Grey()
        textInput.text_horizontal_align = nanome.ui.TextInput.HorizAlignOptions.Right
        textInput.padding_right = .2
        textInput.text_size = .6

        Logs.message("Added text input")

        ln_list.sizing_type = nanome.ui.LayoutNode.SizingTypes.ratio
        ln_list.sizing_value = 0.5
        ln_list.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_list.padding = (0.01, 0.01, 0.01, 0.01)
        ln_list.forward_dist = .03

        prefab = nanome.ui.LayoutNode()
        prefab.layout_orientation = nanome.ui.LayoutNode.LayoutTypes.vertical
        child1 = nanome.ui.LayoutNode()
        child1.sizing_type = nanome.ui.LayoutNode.SizingTypes.ratio
        child1.sizing_value = .3
        child1.name = "label"
        child1.forward_dist = .01
        child2 = nanome.ui.LayoutNode()
        child2.name = "button"
        child2.forward_dist =.01
        prefab.add_child(child1)
        prefab.add_child(child2)
        prefabLabel = nanome.ui.Label()
        prefabLabel.text_value = "Molecule Label"
        prefabButton = nanome.ui.Button()
        prefabButton.text.active = True
        prefabButton.text.value.set_all("Molecule Button")
        prefabButton.disable_on_press = True
        prefabButton.register_pressed_callback(self.select_button_callback)
```

(continues on next page)

(continued from previous page)

```
        child1.set_content(prefabLabel)
        child2.set_content(prefabButton)

        list_content = []
        for i in range(0, 10):
            clone = prefab.clone()
            list_content.append(clone)

        list = nanome.ui.UIList()
        list.display_columns = 1
        list.display_rows = 1
        list.total_columns = 1
        list.items = list_content

        Logs.message("Added list")

        content.add_child(ln_contentBase)
        ln_contentBase.add_child(ln_label)
        ln_contentBase.add_child(ln_button)
        ln_contentBase.add_child(ln_slider)
        ln_contentBase.add_child(ln_textInput)
        ln_contentBase.add_child(ln_list)
        ln_label.set_content(label)
        ln_button.set_content(button)
        ln_slider.set_content(slider)
        ln_textInput.set_content(textInput)
        ln_list.set_content(list)
        return content

    def create_tab2(self):
        content = nanome.ui.LayoutNode()
        ln_contentBase = nanome.ui.LayoutNode()
        ln_label = nanome.ui.LayoutNode()
        ln_button = nanome.ui.LayoutNode()
        ln_dropdown = nanome.ui.LayoutNode()
        ln_textInput = nanome.ui.LayoutNode()

        content.forward_dist = .02
        content.layer = 1

        ln_label.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_label.padding = (0.01, 0.01, 0.01, 0.01)
        ln_label.forward_dist = .001

        label = nanome.ui.Label()
        label.text_value = "Press the button..."
        label.text_color = nanome.util.Color.White()

        Logs.message("Added Label")

        ln_button.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_button.padding = (0.01, 0.01, 0.01, 0.01)
        ln_button.forward_dist = .001

        button = ln_button.add_new_toggle_switch("Toggle Switch")
        button.text.size = .5
        button.text.auto_size = False
```

(continues on next page)

(continued from previous page)

```
        button.register_pressed_callback(self.loading_bar_callback)
        button.register_hover_callback(self.hover_callback)

        Logs.message("Added button")

        ln_dropdown.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_dropdown.padding = (0.01, 0.01, 0.01, 0.01)
        ln_dropdown.forward_dist = .004

        dropdown = nanome.ui.Dropdown()
        dropdown.items = [nanome.ui.DropdownItem(name) for name in ["option1",
↪"option2", "option3", "option4", "option5", "option6"]]
        dropdown.register_item_clicked_callback(dropdown_callback)

        Logs.message("Added dropdown")

        ln_textInput.padding_type = nanome.ui.LayoutNode.PaddingTypes.ratio
        ln_textInput.padding = (0.01, 0.01, 0.01, 0.01)
        ln_textInput.forward_dist = .001

        textInput = nanome.ui.TextInput()
        textInput.max_length = 30
        textInput.register_changed_callback(text_changed_callback)
        textInput.register_submitted_callback(text_submitted_callback)
        textInput.password = True
        textInput.input_text = "hello"

        Logs.message("Added text input")

        prefab = nanome.ui.LayoutNode()
        prefab.layout_orientation = nanome.ui.LayoutNode.LayoutTypes.vertical
        child1 = nanome.ui.LayoutNode()
        child1.sizing_type = nanome.ui.LayoutNode.SizingTypes.ratio
        child1.sizing_value = .3
        child1.name = "label"
        child1.forward_dist = .01
        child2 = nanome.ui.LayoutNode()
        child2.name = "button"
        child2.forward_dist =.01
        prefab.add_child(child1)
        prefab.add_child(child2)
        prefabLabel = nanome.ui.Label()
        prefabLabel.text_value = "Molecule Label"
        prefabButton = nanome.ui.Button()
        prefabButton.text.active = True
        prefabButton.text.value.set_all("Molecule Button")
        prefabButton.register_pressed_callback(self.select_button_callback)
        child1.set_content(prefabLabel)
        child2.set_content(prefabButton)

        ln_loading_bar = nanome.ui.LayoutNode(name="LoadingBar")
        ln_loading_bar.forward_dist = .003
        self.loadingBar = ln_loading_bar.add_new_loading_bar()

        content.add_child(ln_contentBase)
        ln_contentBase.add_child(ln_label)
        ln_contentBase.add_child(ln_button)
```

(continues on next page)

(continued from previous page)

```
        ln_contentBase.add_child(ln_dropdown)
        ln_contentBase.add_child(ln_textInput)
        ln_contentBase.add_child(ln_loading_bar)
        ln_label.set_content(label)
        ln_button.set_content(button)
        ln_dropdown.set_content(dropdown)
        ln_textInput.set_content(textInput)
        return content

    def create_tab_buttons(self):
        LN = nanome.ui.LayoutNode
        ln = LN()
        ln.layout_orientation = nanome.util.enums.LayoutTypes.horizontal
        ln._sizing_type = nanome.util.enums.SizingTypes.fixed
        ln._sizing_value = .1

        def tab1_callback(button):
            self.tab_button1.selected  = True
            self.tab_button2.selected  = False
            self.tab1.enabled = True
            self.tab2.enabled = False

            self.update_node(self.tabs)
            self.update_content(self.tab_button1, self.tab_button2)

        def tab2_callback(button):
            self.tab_button2.selected  = True
            self.tab_button1.selected  = False
            self.tab2.enabled = True
            self.tab1.enabled = False

            self.update_node(self.tabs)
            self.update_content([self.tab_button2, self.tab_button1])

        tab_button_node1 = ln.create_child_node("tab1")
        self.tab_button1 = tab_button_node1.add_new_button("tab1")
        self.tab_button1.register_pressed_callback(tab1_callback)
        tab_button_node2 = ln.create_child_node("tab2")
        self.tab_button2 = tab_button_node2.add_new_button("tab2")
        self.tab_button2.register_pressed_callback(tab2_callback)
        return ln

    def __init__(self):
        pass

permissions = [nanome.util.enums.Permissions.local_files_access]
integrations = [nanome.util.enums.Integrations.minimization, nanome.util.enums.
↪Integrations.structure_prep]

nanome.Plugin.setup(NAME, DESCRIPTION, CATEGORY, HAS_ADVANCED_OPTIONS, UIPlugin,␣
↪permissions=permissions, integrations=integrations)
```

1.4.3 Z-fighting problem

A known problem, called z-fighting, is the following:

If you look closely, you will see that the text intersects with its background. This happens when two objects are exactly on the same plan.

To fix this issue, try to set the *forward_dist* of your foreground element (here, the text)

1.5 Files API

Here is a simple example of File API usage, requesting directory and files, and writing files on Nanome machine.

```
import nanome

class FilesAPITest(nanome.PluginInstance):
    def on_run(self):
        self.request_directory(".", self.on_directory_received) # Request all content␣
↪of "." directory (where Nanome is installed)

    def on_directory_received(self, result):
        if result.error_code != nanome.util.DirectoryErrorCode.no_error: # If API␣
↪couldn't access directory, display error
            nanome.util.Logs.error("Directory request error:", str(result.error_code))
            return

        # For each entry in directory, display name and if directory
        for entry in result.entry_array:
            nanome.util.Logs.debug(entry.name, "Is Directory?", entry.is_directory)

        self.request_files(["./api_bad_test.txt", "api_test.txt"], self.on_files_
↪received) # Read two files

    def on_files_received(self, file_list):
```

(continues on next page)

(continued from previous page)

```
        # For each file we read, display if error, and file content
        for file in file_list:
            nanome.util.Logs.debug("Error?", str(nanome.util.FileErrorCode(file.error_
↪code)), "Content:", file.data)

        # Prepare to write file "api_test.txt", with content "AAAA"
        file = nanome.util.FileSaveData()
        file.path = "./api_test.txt"
        file.write_text("AAAA")
        self.save_files([file], self.on_save_files_result) # Write file

    def on_save_files_result(self, result_list):
        # Check for writing errors
        for result in result_list:
            nanome.util.Logs.debug("Saving", result.path, "Error?", str(nanome.util.
↪FileErrorCode(result.error_code)))

if __name__ == "__main__":
    plugin = nanome.Plugin("Example File API", "Test File API by reading current␣
↪directory, reading api_test.txt and api_bad_test.txt and modifying api_test.txt",
↪"Examples", False)
    plugin.set_plugin_class(FilesAPITest)
    plugin.run()
```

1.6 Notifications API

1.6.1 Send a Notification of each type to the user

```
def on_run(self):
    self.send_notification(nanome.util.enums.NotificationTypes.error, "There was an␣
↪error")
    self.send_notification(nanome.util.enums.NotificationTypes.message, "This is a␣
↪message for the user")
    self.send_notification(nanome.util.enums.NotificationTypes.success, "Something␣
↪good might have happened)
    self.send_notification(nanome.util.enums.NotificationTypes.warning, "Something␣
↪bad might have happened")
```

1.7 Class reference

When importing nanome, the “api” module is flattened, meaning that plugin_instance can be referred to as “nanome.plugin_instance” or “nanome.api.plugin_instance”

1.7.1 nanome

nanome package

Subpackages

nanome.api package

Subpackages

nanome.api.integration package

Submodules

nanome.api.integration.integration module

class Integration
Bases: object

nanome.api.integration.integration_request module

class IntegrationRequest(*request_id*, *type*, *args*, *network*)
Bases: object

get_args()

send_response(*args*)

nanome.api.macro package

Submodules

nanome.api.macro.macro module

class Macro(*title=''*, *logic=''*)
Bases: nanome._internal._macro._macro._Macro

delete(*all_users=False*)

classmethod get_live(*callback=None*)

classmethod get_plugin_identifier()

logic

run(*callback=None*)

save(*all_users=False*)

classmethod set_plugin_identifier(*value*)

classmethod stop()

title

nanome.api.shapes package

Submodules

nanome.api.shapes.anchor module

class Anchor
Bases: nanome._internal._shapes._anchor._Anchor

anchor_type

global_offset

local_offset

target

viewer_offset

nanome.api.shapes.label module

class Label
Bases: nanome._internal._shapes._label._Label, *nanome.api.shapes.shape.Shape*

anchors

Anchors of the shape

Parameters **value** (list of Anchor) – Anchors of the shape

font_size

text

nanome.api.shapes.line module

class Line
Bases: nanome._internal._shapes._line._Line, *nanome.api.shapes.shape.Shape*

anchors

Anchors of the shape

Parameters **value** (list of Anchor) – Anchors of the shape

dash_distance

dash_length

thickness

nanome.api.shapes.shape module

class Shape(*shape_type*)
Bases: nanome._internal._shapes._shape._Shape

Base class of a shape. Used in self.create_shape(shape_type) in plugins.

Parameters **shape_type** (*ShapeType*) – Enumerator representing the shape_type to create

anchors

Anchors of the shape

Parameters **value** (list of Anchor) – Anchors of the shape

color

Color of the shape

Parameters **value** (Color) – Color of the shape

destroy()

Remove the shape from the Nanome App and destroy it.

index

Index of the shape

shape_type

Type of shape. Currently Sphere, Line, and Label are supported.

Return type *ShapeType*

upload(*done_callback=None*)

Upload the shape to the Nanome App

nanome.api.shapes.sphere module

class Sphere

Bases: nanome._internal._shapes._sphere._Sphere, *nanome.api.shapes.shape.Shape*

Represents a sphere. Can display a sphere in Nanome App.

radius

Radius of the sphere

Parameters **value** (*float*) – Radius of the sphere

nanome.api.streams package

Submodules

nanome.api.streams.stream module

class Stream(*network*, *id*, *data_type*, *direction*)
Bases: object

Class allowing a update or read properties of a lot of structures
Created by calling `create_writing_stream()` or `create_reading_stream()`

When created, a stream is linked to a number of structures. Each call to *update()* will update all these structures

DataType
alias of *nanome.util.enums.StreamDataType*

Direction
alias of *nanome.util.enums.StreamDirection*

Type
alias of *nanome.util.enums.StreamType*

destroy()

Destroy stream once plugin doesn't need it anymore

set_on_interrupt_callback(*callback*)

Sets the function to call if the stream gets interrupted (crash)

set_update_received_callback(*callback*)

Sets the function to call if the stream is reading and received an update

update(*data*, *done_callback=None*)

Send data to the stream, updating all its atoms

Parameters **data** (list of `float` for position and scale streams, list of `byte` for color streams) – List of data to send. i.e, for position stream: x, y, z, x, y, z, etc. (atom 1, atom 2, etc.)

nanome.api.structure package

Subpackages

nanome.api.structure.client package

Submodules

nanome.api.structure.client.workspace_client module

class WorkspaceClient(*base_object=None*)
Bases: nanome._internal._addon._Addon

classmethod compute_hbonds(*callback=None*)

nanome.api.structure.io package

Submodules

nanome.api.structure.io.complex_io module

class ComplexIO(*base_object=None*)
Bases: nanome._internal._addon._Addon

class MMCIFSaveOptions
Bases: object

Options for saving MMCIF files.
Includes options for writing:
- hydrogens
- only selected atoms

class PDBSaveOptions
Bases: object

Options for saving PDB files.
Includes options for writing:
- hydrogens
- TER records
- bonds
- heterogen bonds
- only selected atoms

class SDFSaveOptions
Bases: object

Options for saving SDF files.
Includes options for writing:
- all bonds
- heterogen bonds

from_mmcif(***kwargs*)

Loads the complex from a .cif file

Returns The complex read from the file

Return type Complex

Parameters **kwargs** – See below

Keyword Arguments

path **(str)** Path to the file containing the structure

file **(file)** Opened file containing the structure

lines **(list of str)** List of lines from the file

string **(str)** Contents of the file as a single string

from_pdb(***kwargs*)

Loads the complex from a .pdb file

Returns The complex read from the file

Return type Complex

Parameters **kwargs** – See below

Keyword Arguments

path **(str)** Path to the file containing the structure

file **(file)** Opened file containing the structure

lines **(list of str)** List of lines from the file

string **(str)** Contents of the file as a single string

from_sdf(***kwargs*)

Loads the complex from a .sdf file

Returns The complex read from the file

Return type Complex

Parameters **kwargs** – See below

Keyword Arguments

path **(str)** Path to the file containing the structure

file **(file)** Opened file containing the structure

lines **(list of str)** List of lines from the file

string **(str)** Contents of the file as a single string

to_mmcif(*path*, *options=None*)

Saves the complex into a .cif file

Parameters

- **path** (*str*) – Path to the file
- **options** (*MMCIFSaveOptions*) – Save options

to_pdb(*path*, *options=None*)

Saves the complex into a .pdb file

Parameters

- **path** (*str*) – Path to the file
- **options** (*PDBSaveOptions*) – Save options

to_sdf(*path*, *options=None*)

Saves the complex into a .sdf file

Parameters

- **path** (*str*) – Path to the file
- **options** (*SDFSaveOptions*) – Save options

nanome.api.structure.io.molecule_io module

class MoleculeIO
Bases: object

nanome.api.structure.io.workspace_io module

class WorkspaceIO
Bases: object

Submodules

nanome.api.structure.atom module

class Atom
Bases: nanome._internal._structure._atom._Atom, *nanome.api.structure.base.Base*

Represents an Atom

class AtomRenderingMode
Bases: enum.IntEnum

Shape types an atom can be rendered as.
To be used with atom.atom_mode

Adaptive = 6

BFactor = 5

BallStick = 0

Point = 4

Stick = 1

VanDerWaals = 3

Wire = 2

class Molecular(*parent*)
Bases: object

is_het

name

position

serial

symbol

class Rendering(*parent*)
Bases: object

atom_color

atom_mode

atom_rendering

label_text

labeled

selected

set_visible(*value*)

surface_color

surface_opacity

surface_rendering

acceptor

atom_color

Color of the atom

Type Color

atom_mode

Represents how the atom should be shown, such as ball and point or wired.

Type *AtomRenderingMode*

atom_rendering

Represents if the atom should be rendered specifically.

Type bool

atom_scale

Scale/size/radius of the atom

Type float

bfactor

bonds

Bonds that the atom is part of

chain

Chain that the atom is part of

complex

Complex that the atom is part of

conformer_count

current_conformer

donor

exists

Represents if atom exists for calculations.

Type bool

formal_charge

in_conformer

is_het

Represents if the atom is a HET (Heteroatom - not C or H).

Type bool

label_text

Represents the text that would show up if atom is labeled.

Type str

labeled

Represents if the atom has a label or not. If it does, show the label.

Type bool

molecular

molecule

Molecule that the atom is part of

name

Represents the name of the atom. Ideally, the same as symbol.

Type str

occupancy

partial_charge

position

Position of the atom

Type Vector3

positions

rendering

residue

Residue that the atom is part of

selected

Represents if the atom is currently selected in the Nanome workspace.

Type bool

serial

set_visible(*value*)

Set the atom to be visible or invisible in Nanome.

Type bool

surface_color

Color of the atom surface

Type `Color`

surface_opacity

Opacity of the atom surface

Type `float`

surface_rendering

Represents if the atom surface should be rendered specifically.

Type `bool`

symbol

Represents the symbol of the atom. E.g.: C for Carbon

Type `str`

nanome.api.structure.base module

class Base

Bases: `nanome._internal._structure._base._Base`

Represents the base of a chemical structure (atom, molecule, etc)

index

Index of the base (int)

nanome.api.structure.bond module

class Bond

Bases: `nanome._internal._structure._bond._Bond`, *nanome.api.structure.base.Base*

Represents a Bond between two atoms

class Kind

Bases: `enum.IntEnum`

Bond types.

To be used with bond.kind and elements of bond.kinds

Aromatic = 4

CovalentDouble = 2

CovalentSingle = 1

CovalentTriple = 3

Unknown = 0

class Molecular(*parent*)

Bases: object

kind

atom1

First atom linked by this bond

Type Atom

atom2

Second atom linked by this bond

Type Atom

chain

Chain that the bond is part of

complex

Complex that the bond is part of

conformer_count

current_conformer

exists

Represents if bond exists for calculations.

Type bool

in_conformer

kind

Kind of bond

Type *Kind*

kinds

molecular

molecule

Molecule that the bond is part of

residue

Residue that the bond is part of

nanome.api.structure.chain module

class Chain

Bases: nanome._internal._structure._chain._Chain, *nanome.api.structure.base.Base*

Represents a Chain. Contains residues. Molecules contains chains.

class Molecular(*parent*)

Bases: object

name

add_residue(*residue*)

Add a residue to this chain

Parameters **residue** (Residue) – Residue to add to the chain

atoms

The list of atoms that this chain's residues contain

bonds

The list of bonds within this chain

complex

Complex that this chain is in

molecular

molecule

Molecule that this chain is in

name

Represents the name of the chain

Type str

remove_residue(*residue*)

Remove a residue from this chain

Parameters **residue** (Residue) – Residue to remove from the chain

residues

The list of residues that this chain contains

nanome.api.structure.complex module

class Complex

Bases: nanome._internal._structure._complex._Complex, *nanome.api.structure.base.Base*

Represents a Complex that contains molecules.

class Molecular(*parent*)

Bases: object

index_tag

name

split_tag

class Rendering(*parent*)

Bases: object

box_label

boxed

computing

current_frame

get_selected()

locked

set_surface_needs_redraw()

visible

class Transform(*parent*)

Bases: object

get_complex_to_workspace_matrix()

get_workspace_to_complex_matrix()

position

rotation

add_molecule(*molecule*)

Add a molecule to this complex

Parameters **molecule** (Molecule) – Molecule to add to the chain

static align_origins(*target_complex*, **other_complexes*)

atoms

The list of atoms within this complex

bonds

The list of bonds within this complex

box_label

Represents the label on the box surrounding the complex

Type str

boxed

Represents if this complex is boxed/bordered in Nanome.

Type bool

chains

The list of chains within this complex

computing

convert_to_conformers(*force_conformers=None*)

convert_to_frames()

current_frame

Represents the current animation frame the complex is in.

Type int

full_name

Represents the full name of the complex with its tags and name

Type str

get_all_selected()

get_complex_to_workspace_matrix()

get_selected()

get_workspace_to_complex_matrix()

index_tag

io = <nanome.api.structure.io.complex_io.ComplexIO object>

locked

Represents if this complex is locked and unmovable in Nanome.

Type bool

molecular

molecules

The list of molecules within this complex

name

Represents the name of the complex

Type str

position

Position of the complex

Type Vector3

register_complex_updated_callback(*callback*)

register_selection_changed_callback(*callback*)

remove_molecule(*molecule*)

Remove a molecule from this complex

Parameters **molecule** (Molecule) – Molecule to remove from the chain

rendering

residues

The list of residues within this complex

rotation

Rotation of the complex

Type Quaternion

set_all_selected(*value*)

set_current_frame(*value*)

set_surface_needs_redraw()

split_tag

transform

visible

Represents if this complex is visible in Nanome.

Type bool

nanome.api.structure.molecule module

class Molecule

Bases: nanome._internal._structure._molecule._Molecule, *nanome.api.structure.base.Base*

Represents a molecule. Contains chains.

class Molecular(*parent*)

Bases: object

name

add_chain(*chain*)

Add a chain to this molecule

Parameters **chain** (Chain) – Chain to add to the molecule

associated

Metadata associated with the molecule.
PDB REMARKs end up here.

Type dict

associateds

atoms

The atoms of this complex

Type generator <Atom>

bonds

The bonds of this complex

Type generator <Bond>

chains

The chains of this complex

Type generator <Chain>

complex

Complex that the molecule belongs to

conformer_count

copy_conformer(*src*, *index=None*)

create_conformer(*index*)

current_conformer

delete_conformer(*index*)

molecular

move_conformer(*src*, *dest*)

name

Represents the name of the molecule

Type str

names

remove_chain(*chain*)

Remove a chain from this molecule

Parameters **chain** (Chain) – Chain to remove from the molecule

residues

The residues of this complex

Type generator <Residue>

set_conformer_count(*count*)

set_current_conformer(*index*)

nanome.api.structure.residue module

class Residue

Bases: nanome._internal._structure._residue._Residue, *nanome.api.structure.base.Base*

Represents a Residue. Contains atoms. Chains contain residues.

class Molecular(*parent*)
Bases: object

name

secondary_structure

serial

type

class Rendering(*parent*)
Bases: object

label_text

labeled

ribbon_color

ribbon_mode

ribbon_size

ribboned

class RibbonMode
Bases: enum.IntEnum

Ribbon display modes.
To be used with structure.Residue().ribbon_mode

AdaptiveTube = 1

Coil = 2

SecondaryStructure = 0

class SecondaryStructure
Bases: enum.IntEnum

Secondary structure types.
To be used with structure.Residue().secondary_structure

Coil = 1

Helix = 3

Sheet = 2

Unknown = 0

add_atom(*atom*)

Add an atom to this residue

Parameters **atom** (Atom) – Atom to add to the residue

add_bond(*bond*)

Add a bond to this residue

Parameters **bond** (Bond) – Bond to add to the residue

atoms

The list of atoms within this complex

bonds

The list of bonds within this complex

chain

Chain that the residue is part of

complex

Complex that the residue is part of

label_text

Represents the text that would show up if residue is labeled.

Type str

labeled

Represents if the residue has a label or not. If it does, show the label.

Type bool

molecular

molecule

Molecule that the residue is part of

name

Represents the name of the residue

Type str

remove_atom(*atom*)

Remove an atom from this residue

Parameters **atom** (Atom) – Atom to remove from the residue

remove_bond(*bond*)

Remove a bond from this residue

Parameters **bond** (Bond) – Bond to remove from the residue

rendering

ribbon_color

Color of the ribbon residue

Type Color

ribbon_mode

Represents how the residue ribbon should be shown

Type *RibbonMode*

ribbon_size

ribboned

secondary_structure

The secondary structure of the residue

Type *SecondaryStructure*

serial

type

nanome.api.structure.workspace module

class Workspace

Bases: nanome._internal._structure._workspace._Workspace

Workspace that contains all the complexes shown in Nanome.

class Transform(*parent*)

Bases: object

position

rotation

scale

add_complex(*complex*)

Add complex to the workspace

Parameters **complex** (Complex) – Complex to add to the workspace

client = <nanome.api.structure.client.workspace_client.WorkspaceClient object>

complexes

The list of complexes within the workspace

Type list of Complex

get_workspace_to_world_matrix()

get_world_to_workspace_matrix()

position

Position of the workspace

Type Vector3

remove_complex(*complex*)

Remove complex from the workspace

Parameters **complex** (Complex) – Complex to remove from the workspace

rotation

Rotation of the workspace

Type Quaternion

scale

Scale of the workspace

Type Vector3

transform

nanome.api.ui package

Subpackages

nanome.api.ui.io package

Submodules

nanome.api.ui.io.layout_node_io module

class LayoutNodeIO(*base_object=None*)
Bases: nanome._internal._addon._Addon

A class for json serialization and parsing of LayoutNode objects.

Parameters **base_object** (LayoutNode) – The LayoutNode to serialize

from_json(*path*)

Parses a LayoutNode json file and returns a LayoutNode.

Parameters **path** (str) – The path to the LayoutNode json to parse

to_json(*path*)

Serializes this instance's base_object to the json file specified by path.

Parameters **path** (str) – The path to serialize base_object's json representation to

nanome.api.ui.io.menu_io module

class MenuIO(*base_object=None*)
Bases: nanome._internal._addon._Addon

from_json(*path*)

Parses a Menu json file and returns a Menu.

Parameters **path** (str) – The path to the Menu json to parse

to_json(*path*)

Serializes this instance's base_object to the json file specified by path.

Parameters **path** (str) – The path to serialize base_object's json representation to

update_json(*path*)

Updates a menu written for an old version of the library.
Call once before reading and run once. Then you can remove the call.

Parameters **path** (str) – path to the menu you wish to update.

Submodules

nanome.api.ui.button module

class Button(*text=None*, *icon=None*)
Bases: nanome._internal._ui._button._Button, *nanome.api.ui.ui_base.UIBase*

Represents a clickable button on a menu

class ButtonIcon
Bases: nanome._internal._ui._button._ButtonIcon

active

Whether or not the icon is visible

Type bool

color

The color of the image by button state.

Type MultiStateVariable

position

The position of the icon
A position of (1, 1, 1) represents right, top, front,
whereas (0, 0, 0) represents the middle.

Type tuple<float, float, float>

ratio

The ratio of height to height + width for the icon.
A size of 0.5 represents equal width and height

Type float

rotation

The rotation of the icon about each axis.
A position of (90, 90, 90) represents a quarter rotation
about each of the X, Y and Z axes, whereas (0, 0, 0) represents no rotation.

Type tuple<float, float, float>

sharpness

The sharpness of the icon image (between 0 and 1)

Type float

size

The size of the icon image
A size of 1 represents the full size.

Type float

value

The file paths to the icon image by button state.

Type MultiStateVariable

class ButtonMesh

Bases: nanome._internal._ui._button._ButtonMesh

active

Whether or not the mesh is visible

Type bool

color

The color of the mesh by button state

Type MultiStateVariable

enabled

Whether or not the mesh is visible by button state

Type MultiStateVariable

class ButtonOutline

Bases: nanome._internal._ui._button._ButtonOutline

active

Whether or not the outline is visible

Type bool

color

size

The line thickness of the outline by button state

Type MultiStateVariable

class ButtonSwitch

Bases: nanome._internal._ui._button._ButtonSwitch

active

Whether or not the button is visible

Type bool

off_color

The color for the button when it is off

Type Color

on_color

The color for the button when it is on

Type Color

class ButtonText

Bases: nanome._internal._ui._button._ButtonText

active

Whether or not the button text is visible

Type bool

auto_size

Whether or not to automatically scale the font size of the text
based on the size of the button

Type bool

bold

Whether or not the text is bold by button state

Type MultiStateVariable

color

The color of the text by button state

Type MultiStateVariable

ellipsis

Whether or not to use an ellipsis if there is too much text to display

Type bool

horizontal_align

The horizontal alignment of the text

Type *HorizAlignOptions*

line_spacing

The space between lines of text

Type float

max_size

The maximum font size the text will display
This is the upper bound for auto sizing.

Type float

min_size

The minimum font size the text will display
This is the lower bound for auto sizing.

Type float

padding_bottom

The padding below the text

Type float

padding_left

The padding to the left of the text

Type float

padding_right

The padding to the right of the text

Type float

padding_top

The padding above the text

Type float

size

The font size of the text displayed

Type float

underlined

Whether or not the button text is underlined.

Type bool

value

The text displayed by button state

Type MultiStateVariable

vertical_align

The vertical alignment of the text

Type *VertAlignOptions*

class ButtonTooltip
Bases: nanome._internal._ui._button._ButtonTooltip

bounds

The bounds of the tooltip

Type tuple<float, float, float>

content

The main textual content of the tooltip

Type str

positioning_origin

Determines which part of the tooltip is closest to the button (target)
Refers to the tooltip

Type *ToolTipPositioning*

positioning_target

Determines which side of the button the tooltip (origin) will appear on
Refers to the tooltip's button

Type *ToolTipPositioning*

title

The title of the tooltip

Type str

class HorizAlignOptions
Bases: enum.IntEnum

Horizontal alignment modes for text.
To be used with ui.Label().text_horizontal_align and ui.Button().horizontal_align

Left = 0

Middle = 1

Right = 2

class MultiStateVariable(*default=None*)
Bases: nanome._internal._ui._button._MultiStateVariable

highlighted

Represents the highlighted state where the element is being hovered

Type Any

idle

Represents the idle state where the element is not being hovered and is not selected

Type Any

selected

Represents the highlighted state where the element has been selected

Type Any

selected_highlighted

Represents the selected, highlighted state where the element has been selected and is being hovered over

Type Any

set_all(*value*)

Sets the value for every state

set_each(*idle=None*, *selected=None*, *highlighted=None*, *selected_highlighted=None*, *unusable=None*, *default=None*)

Sets the value for each state

unusable

Represents the unusable state where the element cannot be interacted with

Type Any

class VertAlignOptions

Bases: enum.IntEnum

Vertical alignment modes for text.
To be used with ui.Label().text_vertical_align and ui.Button().vertical_align

Bottom = 2

Middle = 1

Top = 0

disable_on_press

Whether or not to disable the button after it has been pressed once.

Type bool

name

The name of the button

Type str

register_hover_callback (*func*)

Registers a function to be called when the button is hovered over

Parameters **func** (method (Button) -> None) – called when a button is hovered over

register_pressed_callback (*func*)

Registers a function to be called when the button is pressed/clicked

Parameters **func** (method (Button) -> None) – called when a button is pressed

selected

Whether or not the button is selected
Corresponds to a potentially visually distinct UI state

Type bool

toggle_on_press

Whether or not to toggle the selected state of the button when it is pressed.

Type bool

unusable

Whether or not the button is unusable
Corresponds to a potentially visually distinct UI state

Type bool

nanome.api.ui.dropdown module

class Dropdown

Bases: nanome._internal._ui._dropdown._Dropdown, *nanome.api.ui.ui_base.UIBase*

Represents a dropdown menu

items

A list of DropdownItems in the list

Type list <DropdownItem>

max_displayed_items

The maximum number of items to display at a time
If there are more items in the dropdown than this value,
a scrollbar will be appear on the dropdown.

Type int

permanent_title

The permanent text to display over the Dropdown's selected item area.

Type str

register_item_clicked_callback(*func*)

Registers a function to be called when a dropdown item is pressed

Parameters **func** (method (Dropdown, DropdownItem) -> None) – called when a dropdown item is pressed

use_permanent_title

Whether or not to display permanent text where the Dropdown would otherwise display the selected item

Type bool

nanome.api.ui.dropdown_item module

class DropdownItem(*name='item'*)
Bases: nanome._internal._ui._dropdown_item._DropdownItem

Represents a dropdown item in a dropdown menu

clone()

Returns a deep copy this DropdownItem.

Type DropdownItem

close_on_selected

Whether or not this item will close the Dropdown after being selected
Setting this value to false can allows for multiple items to be selected.

Type bool

name

The name of the Dropdown item.
This text is displayed on the item when the Dropdown expands
and in the collapsed Dropdown when the item is the selected item.

Type str

selected

Whether or not this item is selected.
In the case that a single DropdownItem is selected in a Dropdown,
the item's text will appear on the Dropdown when it is collapsed

Type bool

nanome.api.ui.image module

class Image(*file_path=''*)
Bases: nanome._internal._ui._image._Image, *nanome.api.ui.ui_base.UIBase*

Represents an image in a menu

class ScalingOptions
Bases: enum.IntEnum

Ways for an image to scale.
To be used with ui.Image().scaling_option

fill = 1

fit = 2

stretch = 0

color

The color of the image

Type Color

file_path

The file path to the image.
Setting this and calling update_content will change the image.

Type str

register_held_callback(*func*)

Registers a function to be called rapidly while the image is being pressed

Parameters **func** (method (Image, int, int) -> None) – called while the image is being pressed

register_pressed_callback(*func*)

Registers a function to be called when the image is pressed

Parameters **func** (method (Image, int, int) -> None) – called the image is pressed

register_released_callback(*func*)

Registers a function to be called when the image is released

Parameters **func** (method (Image, int, int) -> None) – called the image is released

scaling_option

Determines how the image scales.

Type *ScalingOptions*

nanome.api.ui.label module

class Label(*text=None*)

Bases: nanome._internal._ui._label._Label, *nanome.api.ui.ui_base.UIBase*

Represents a label that cannot be interacted with in a menu

class HorizAlignOptions

Bases: enum.IntEnum

Horizontal alignment modes for text.
To be used with ui.Label().text_horizontal_align and ui.Button().horizontal_align

Left = 0

Middle = 1

Right = 2

class VertAlignOptions

Bases: enum.IntEnum

Vertical alignment modes for text.
To be used with ui.Label().text_vertical_align and ui.Button().vertical_align

Bottom = 2

Middle = 1

Top = 0

text_auto_size

Whether or not to automatically size the label text

Type bool

text_bold

Whether or not the text on this label is bold

Type bool

text_color

The color of the text on this label

Type Color

text_horizontal_align

The horizontal alignment of the text

Type *HorizAlignOptions*

text_italic

Whether or not the text on this label is italic

Type bool

text_max_size

The maximum font size the text will display
This is the upper bound for auto sizing.

Type float

text_min_size

The minimum font size the text will display
This is the lower bound for auto sizing.

Type float

text_size

The font size of the text displayed on this label

Type float

text_underlined

Whether or not the text on this label is underlined

Type bool

text_value

The text to be displayed on the label

Type str

text_vertical_align

The vertical alignment of the text

Type *VertAlignOptions*

nanome.api.ui.layout_node module

class LayoutNode(*name='node'*)
Bases: nanome._internal._ui._layout_node._LayoutNode

Class for hierarchical UI objects representing part of a Nanome menu.
Layout nodes are used to create menus, by defining where one UI element should be placed relative to another.
One LayoutNode can contain one interactive UI element as well as any number of child Layout Nodes.

Parameters **name** (str) – Name of the node, used to identify it and find it later

class LayoutTypes
Bases: enum.IntEnum

Orientation modes for Layout Nodes.
To be used with ui.LayoutNode().layout_orientation

horizontal = 1

vertical = 0

class PaddingTypes
Bases: enum.IntEnum

UI padding types.
To be used with ui.LayoutNode().padding_type

fixed = 0

ratio = 1

class SizingTypes
Bases: enum.IntEnum

Ways in which a Layout Node can be sized within a UI layout.
To be used with ui.LayoutNode().sizing_type

expand = 0

fixed = 1

ratio = 2

add_child(*child_node*)

add_new_button(*text=None*)

add_new_dropdown()

add_new_image(*file_path=''*)

add_new_label(*text=None*)

add_new_list()

add_new_loading_bar()

add_new_mesh()

add_new_slider(*min_value=0*, *max_value=10*, *current_value=5*)

add_new_text_input(*placeholder_text=''*)

add_new_toggle_switch(*text=None*)

clear_children()

clone()

create_child_node(*name=''*)

enabled

Defines if layout node is visible.
If disabled, it will not influence the menu layout.

Type bool

find_ancestor(*name*)

find_node(*name*, *recursively=True*)

Checks child nodes for a node of the matching name.
If "recursively" is True, this also checks all descending nodes.

Parameters **name** (*str*) – Name of the node to find.

Returns LayoutNode with matching name

Return type LayoutNode

forward_dist

Sets the depth distance (towards camera) of a node, relative to its parent

Type float

get_children()

get_content()

io = <nanome.api.ui.io.layout_node_io.LayoutNodeIO object>

layer

The node layer. A node on layer 0 and another on layer 1 will be on different layouts, possibly overlapping

Type int

layout_orientation

Defines if children node should be arranged vertically or horizontally

Type LayoutOrientation

name

Name of the node, used to identify it and find it later

Type str

padding

padding_type

The padding type of the LayoutNode.

Type *PaddingTypes*

parent

remove_child(*child_node*)

remove_content()

set_content(*ui_content*)

set_padding(*left=None*, *right=None*, *top=None*, *down=None*)

set_size_expand()

set_size_fixed(*size*)

set_size_ratio(*size*)

sizing_type

Defines how the node size in the layout should be calculated

Type *SizingTypes*

sizing_value

Size of the node in its layout.
Behavior is different depending of *sizing_type*

Type float

nanome.api.ui.loading_bar module

class LoadingBar

Bases: nanome._internal._ui._loading_bar._LoadingBar, *nanome.api.ui.ui_base.UIBase*

Represents a loading bar that can display a percentage

description

A description of what is being loaded.
Appears under the loading bar title

Type str

failure

Whether or not loading has failed
Setting this to true and updating the UI will make the loading bar appear red in Nanome

Type bool

percentage

The load percentage to indicate

Type float

title

The title of the loading bar.
Appears over the loading bar

Type str

nanome.api.ui.menu module

class Menu(*index=0*, *title='title'*)
Bases: nanome._internal._ui._menu._Menu

Represents a menu for a plugin

enabled

Determines the visibility of the menu

Type bool

find_content(*content_id*)

Finds a piece of content by its content ID.

Parameters **content_id** (int) – the ID of the content to find

Returns The UI content on this menu matching the ID

Return type UIBase

get_all_content()

Gets all content from this menu

Returns A list of all UI content on this menu

Return type list <UIBase>

get_all_nodes()

Gets all LayoutNodes from this menu

Returns A list of all LayoutNodes on this menu

Return type list <LayoutNode>

height

The height of the menu

Type float

index

The index of the menu.
Used to determine a menu's identity.
Menus with the same index will replace one another when updated.

Type int

io = <nanome.api.ui.io.menu_io.MenuIO object>

locked

Whether or not the menu is locked in place

Type bool

register_closed_callback(*func*)

Registers a function to be called when the menu's close button is pressed.

Parameters **func** (method (Menu) -> None) – called the menu is closed

root

The hierarchical root LayoutNode of the menu

Type LayoutNode

title

The title which appears at the top of the menu

Type str

width

The width of the menu

Type float

nanome.api.ui.mesh module

class Mesh

Bases: nanome._internal._ui._mesh._Mesh, *nanome.api.ui.ui_base.UIBase*

Represents a flat rectangular mesh with a solid color.

mesh_color

The color of the mesh

Type Color

nanome.api.ui.slider module

class Slider(*min_val=None*, *max_val=None*, *current_val=None*)
Bases: nanome._internal._ui._slider._Slider, *nanome.api.ui.ui_base.UIBase*

Represents a slider that has a set range of values

current_value

The current value of the slider

Type float

max_value

The minimum (far right) value of the slider

Type float

min_value

The minimum (far left) value of the slider

Type float

register_changed_callback(*func*)

Register a function to be called every time the value of the slider changes

Parameters **func** (method (Slider) -> None) – callback function to execute when slider changes values

register_released_callback(*func*)

Register a function to be called when the slider is released.

Parameters **func** (method (Slider) -> None) – callback function to execute when slider is released

nanome.api.ui.text_input module

class TextInput

Bases: nanome._internal._ui._text_input._TextInput, *nanome.api.ui.ui_base.UIBase*

Represents a text input, where the user can input text

class HorizAlignOptions

Bases: enum.IntEnum

Horizontal alignment modes for text.
To be used with ui.Label().text_horizontal_align and ui.Button().horizontal_align

Left = 0

Middle = 1

Right = 2

background_color

The color of the background of this text input

Type Color

input_text

The string that has been entered into this text input

Type str

max_length

The character limit of the input string

Type int

number

Whether or not the input represents a number.
Will display the number keyboard if set to true.

Type bool

padding_bottom

The bottom padding of the input and placeholder text

Type float

padding_left

The left padding of the input and placeholder text

Type float

padding_right

The right padding of the input and placeholder text

Type float

padding_top

The top padding of the input and placeholder text

Type float

password

Whether or not the input represents a password.
i.e. will display 123 as ***** if true.

Type bool

placeholder_text

The text to display when the input is empty

Type str

placeholder_text_color

Color of the placeholder text

Type Color

register_changed_callback(*func*)

Registers a function to be called whenever the text input is changed.
The function must take a text input as its only parameter.

Parameters **func** – The function that will be called when the text input is changed.

register_submitted_callback(*func*)

Registers a function to be called whenever the user submits a text input.

The function must take a text input as its only parameter.

Parameters **func** – The function that will be called when the user submits a text input.

text_color

The color of the input text

Type Color

text_horizontal_align

The horizontal alignment of the input and placeholder text

Type *HorizAlignOptions*

text_size

The font size of the input and placeholder text

Type float

nanome.api.ui.ui_base module

class UIBase

Bases: object

clone()

nanome.api.ui.ui_list module

class UIList

Bases: nanome._internal._ui._ui_list._UIList, *nanome.api.ui.ui_base.UIBase*

A class representing a list of UI elements.

display_columns

Number of columns of items to display simultaneously.

Type int

display_rows

Number of rows of items to display simultaneously.

Type int

items

LayoutNodes items to be displayed in the list.

Type list <LayoutNode>

total_columns

Total number of columns to display across scrolling.
i.e. If there are 2 display columns and 4 total columns,
the horizontal scroll bar will have two possible positions.

Type int

unusable

Whether or not the UI list is usable.

Type bool

nanome.api.user package

Submodules

nanome.api.user.presenter_info module

class PresenterInfo
Bases: object

Class to fetch information about the current nanome session's presenter.

account_email

The Nanome account email of the presenter

Type str

account_id

The Nanome account ID of the presenter

Type str

account_name

The Nanome account name of the presenter

Type str

Submodules

nanome.api.files module

class Files(*plugin_instance*)
Bases: nanome._internal._files._Files

Class to navigate through files and directories on the machine running Nanome using unix-like filesystem methods.

cd(*directory*, *callback=None*)

changes the current working directory

Parameters

- **directory** (*str*) – directory to change to
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

cp(*source*, *dest*, *callback=None*)

Copy source to dest

Parameters

- **source** (*str*) – the Nanome machine filename of the file to copy
- **dest** (*str*) – the Nanome machine filename to copy to
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

get(*source*, *dest*, *callback=None*)

Gets file source from the Nanome session's machine and writes to dest of the plugin machine.

Parameters

- **source** (*str*) – Nanome machine filename of the file to move
- **dest** (*str*) – plugin machine filename for the file's destination
- **callback** (method (*FileError*, str) -> None) – called when operation has completed, with dest and any potential errors

ls(*directory*, *callback=None*)

list directory's contents

Parameters

- **directory** (*str*) – directory to request
- **callback** (method (*FileError*, list of *FileMeta*) -> None) – function that will be called with contents of the directory

mkdir(*target*, *callback=None*)

Create all directories along the path provided

Parameters

- **target** (*str*) – pathname of the final directory to create
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

mv(*source*, *dest*, *callback=None*)

Rename source to dest, or move source into directory dest/

Parameters

- **source** (*str*) – file to move or rename
- **dest** (*str*) – file or pathname of the file's destination
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

put(*source*, *dest*, *callback=None*)

Send the file source on the plugin machine to be placed at dest on the Nanome session's machine.

Parameters

- **source** (*str*) – plugin machine filename of the file to send
- **dest** (*str*) – Nanome machine filename for the file's destination
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

pwd(*callback=None*)

Print the absolute path of the current working directory

Parameters **callback** (method (*FileError*, str) -> None) – function that will be called with the full working directory path

rm(*target*, *callback=None*)

remove non-directory file

Parameters

- **target** (*str*) – filepath of Nanome machine file to remove.
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

rmdir (*target*, *callback=None*)

remove directory

Parameters

- **target** (*str*) – Nanome machine directory to remove.
- **callback** (method (*FileError*) -> None) – called when operation has completed, potentially with errors

nanome.api.plugin module

class Plugin (*name*, *description*, *tags=[]*, *has_advanced=False*, *permissions=[]*, *integrations=[]*)
Bases: `nanome._internal._plugin._Plugin`

Core class of any Plugin.
Manages network, callbacks and APIs

Parameters

- **name** (`str`) – Name of the plugin to display
- **description** (`str`) – Description of the plugin to display
- **tags** (`list <str>`) – Tags of the plugin
- **has_advanced** (`bool`) – If true, plugin will display an "Advanced Settings" button

post_run

Function to call when the plugin is about to exit
Useful when using autoreload

pre_run

Function to call before the plugin runs and tries to connect to NTS
Useful when using autoreload

run (*host='config'*, *port='config'*, *key='config'*)

Starts the plugin by connecting to the server specified.
If arguments (-a, -p) are given when starting plugin, host/port will be ignored.
Function will return only when plugin exits.

Parameters

- **host** (*str*) – NTS IP address if plugin started without -a option

- **port** (*int*) – NTS port if plugin started without -p option

static set_custom_data(**args*)

Store arbitrary data to send to plugin instances

Parameters **args** (*Anything serializable*) – Variable length argument list

static set_maximum_processes_count(*max_process_nb*)

set_plugin_class(*plugin_class*)

Set plugin class to instantiate when a new session is connected
The plugin class should interact with or override functions in `PluginInstance` to interact with Nanome

Parameters **plugin_class** (`PluginInstance`) – Plugin class to instantiate

classmethod setup(*name*, *description*, *tags*, *has_advanced*, *plugin_class*, *host='config'*, *port='config'*, *key='config'*, *permissions=[]*, *integrations=[]*)

nanome.api.plugin_instance module

class AsyncPluginInstance

Bases: *nanome.api.plugin_instance.PluginInstance*

Base class of any asynchronous plugin.
Constructor should never be called by the user as it is network-instantiated when a session connects.
All methods available to PluginInstance are available to AsyncPluginInstance.
Decorating these methods with @async_callback will allow them to use the async keyword in their definition

is_async = True

class PluginInstance

Bases: `nanome._internal._plugin_instance._PluginInstance`

Base class of any plugin.
Constructor should never be called by the user as it is network-instantiated when a session connects.
Start, update, and all methods starting by "on" can be overridden by user, in order to get requests results

add_bonds(*complex_list*, *callback=None*, *fast_mode=None*)

Calculate bonds
Requires openbabel to be installed

Parameters

- **complex_list** (list of `Complex`) – List of complexes to add bonds to

- **callback** – Callable[[List[Complex]], None]

add_dssp(*complex_list*, *callback=None*)

Use DSSP to calculate secondary structures

Parameters

- **complex_list** (list of `Complex`) – List of complexes to add ribbons to
- **callback** – Callable[[List[Complex]], None]

add_to_workspace(*complex_list*, *callback=None*)

Add a list of complexes to the current workspace

Parameters **complex_list** (list of `Complex`) – List of Complexes to add

add_volume(*complex*, *volume*, *properties*, *complex_to_align_index=-1*, *callback=None*)

apply_color_scheme(*color_scheme*, *target*, *only_carbons*)

Applies a color scheme to selected atoms.

Parameters

- **color_scheme** (*ColorScheme*) – the color scheme to use on atoms
- **target** (*ColorSchemeTarget*) – whether you want to color the atom, the surface, or the ribbon
- **only_carbons** (*bool*) – whether you want to only color carbons, or all atoms.

center_on_structures(*structures*, *callback=None*)

Repositions the workspace such that the provided structure(s) will be in the center of the world.

Parameters

- **structures** (list of `Base`) – Molecular structure(s) to update.
- **callback** – Callable[[], None]

create_atom_stream(*atom_indices_list*, *stream_type*, *callback*)

create_reading_stream(*indices_list*, *stream_type*, *callback=None*)

Create a stream allowing the plugin to continuously receive properties of many objects

Parameters

- **indices_list** (list of `int`) – List of indices of all objects that should be in the stream
- **stream_type** (list of `Type`) – Type of stream to create
- **callable** – Callable[[Stream, StreamCreationError], None]

create_stream(*atom_indices_list*, *callback*)

create_writing_stream(*indices_list*, *stream_type*, *callback=None*)

Create a stream allowing the plugin to continuously update properties of many objects

Parameters

- **indices_list** (list of `int`) – List of indices of all objects that should be in the stream
- **stream_type** (list of `Type`) – Type of stream to create
- **callback** – Callable[[Stream, StreamCreationError], None]

custom_data

Get custom data set with Plugin.set_custom_data

Type tuple of objects or None if no data has been set

is_async = False

menu

on_advanced_settings()

Called when user presses "Advanced Settings"

on_complex_added()

Called whenever a complex is added to the workspace.

on_complex_removed()

Called whenever a complex is removed from the workspace.

on_presenter_change()

Called when room's presenter changes.

on_run()

Called when user presses "Run"

on_stop()

Called when user disconnects or plugin crashes

open_url(*url*)

Opens a URL alongside the Nanome session in the default web browser.

Parameters **url** (*str*) – url to open

plugin_files_path

request_complex_list(*callback=None*)

Request the list of all complexes in the workspace, in shallow mode

kwarg callback: Callable[[List[Complex]], None]

request_complexes(*id_list*, *callback=None*)

Requests a list of complexes by their indices
Complexes returned contains the full structure (atom/bond/residue/chain/molecule)

Parameters **id_list** (list of int) – List of indices

Callback Callable[[List[Complex]], None]

request_controller_transforms(*callback=None*)

Requests presenter controller info (head position, head rotation, left controller position, left controller rotation, right controller position, right controller rotation)

param callback: Callable[[Vector3, Quaternion, Vector3, Quaternion, Vector3, Quaternion], None]

request_export(*format*, *callback=None*, *entities=None*)
Request a file export using Nanome exporters Can request either molecule or workspace export, for entities in Nanome workspace or directly sent by the plugin (without begin uploaded to workspace)

Parameters

- **format** (*ExportFormats*) – File format to export
- **entities** (list of or unique object of type Workspace or Complex, or None, or list of or unique int) – Entities to export (complexes to send, or indices if referencing complexes in workspace, or a workspace, or nothing if exporting Nanome workspace)
- **callback** – Callable[[Union[str, bytes]], None]

request_menu_transform(*index*, *callback=None*)

Requests spatial information of the plugin menu (position, rotation, scale)

Parameters **index** (*int*) – Index of the menu you wish to read

callback: Callable[[Vector3, Quaternion, Vector3], None]

request_presenter_info(*callback=None*)

Requests presenter account info (unique ID, name, email)

callback: Callable[[PresenterInfo], None]

request_workspace(*callback=None*)

Request the entire workspace, in deep mode

callback: Callable[[Workspace], None]

save_files(*file_list*, *callback=None*)

Save files on the machine running Nanome, and returns result

Parameters

- **file_list** (list of *FileSaveData*) – List of files to save with their content
- **callable** – Callable[[List[FileSaveData]], None]

send_files_to_load(*files_list*, *callback=None*)

Send file(s) to Nanome to load directly using Nanome's importers.
Can send just a list of paths, or a list of tuples containing (path, name)

Parameters **files_list** (list of or unique object of type `str` or (`str`, `str`)) – List of files to load

send_notification(*type*, *message*)

Send a notification to the user

Parameters

- **type** – Type of notification to send.
- **message** (*str*) – Text to display to the user.

set_menu_transform(*index*, *position*, *rotation*, *scale*)

Update the position, scale, and rotation of the menu

Parameters

- **index** (*int*) – Index of the menu you wish to update
- **position** (*vector3*) – New position of the menu
- **rotation** (*quaternion*) – New rotation of the menu
- **scale** (*vector3*) – New scale of the menu

set_plugin_list_button(*button*, *text=None*, *usable=None*)

Set text and/or usable state of the buttons on the plugin connection menu in Nanome

Parameters

- **button** (`ButtonType`) – Button to set
- **text** (*str*) – Text displayed on the button. If None, doesn't set text
- **usable** (*bool*) – Set button to be usable or not. If None, doesn't set usable text

start()

Called when user "Activates" the plugin

update()

Called when when instance updates (multiple times per second)

update_content(**content*)

Update specific UI elements (button, slider, list...)

Parameters **content** (UIBase or multiple UIBase or a list of UIBase) – UI elements to update

update_menu(*menu*)

Update the menu in Nanome

Parameters **menu** (Menu) – Menu to update

update_node(**nodes*)

Updates layout nodes and their children

Parameters **nodes** (LayoutNode or multiple LayoutNode or a list of LayoutNode) – Layout nodes to update

update_structures_deep(*structures*, *callback=None*)

Update the specific molecular structures in the scene to match the structures in parameter.
Will also update descendent structures and can be used to remove descendent structures.

Parameters **structures** (list of Base) – List of molecular structures to update.

callback: Callable[[], None]

update_structures_shallow(*structures*)

Update the specific molecular structures in the scene to match the structures in parameter
Only updates the structure's data, will not update children or other descendents.

Parameters **structures** (list of Base) – List of molecular structures to update.

update_workspace(*workspace*)

Replace the current workspace in the scene by the workspace in parameter

Parameters **workspace** (Workspace) – New workspace

zoom_on_structures(*structures*, *callback=None*)

Repositions and resizes the workspace such that the provided structure(s) will be in the center of the users view.

Parameters

- **structures** (list of Base) – Molecular structure(s) to update.

- **callback** – Callable[[], None]

nanome.api.room module

class Room
Bases: nanome._internal._room._Room

Represents a room in Nanome

class SkyBoxes
Bases: enum.IntEnum

Preset skyboxes to show in a Nanome room
To be used with plugin_instance.room.set_skybox

Black = 3

BlueSkyAndClouds = 0

BlueSkyAndGround = 2

Graydient = 5

Sunset = 1

Unknown = -1

White = 4

set_skybox(*skybox*)

nanome.util package

Submodules

nanome.util.asyncio module

async_callback(*fn*)

nanome.util.color module

class Color(*r=0*, *g=0*, *b=0*, *a=255*, *whole_num=None*)
Bases: object

Represents a 32-bit color with red, green, blue and alpha channels (8 bits each).

Parameters

- **r** (int) – Red component

- **g** (int) – Green component
- **b** (int) – Blue component
- **a** (int) – Alpha component
- **whole_num** (int or hex) – Optional way to input color. The int or hex form of the color. e.g. 0x8000FFFF

classmethod Black()

classmethod Blue()

classmethod Clear()

classmethod Gray()

classmethod Green()

classmethod Grey()

classmethod Red()

classmethod White()

classmethod Yellow()

a

The alpha component of the color.

Type int

b

The blue component of the color.

Type int

copy()

Create a new color from this one.

Return type Color

classmethod from_int(*value*)

Set color from int after initializing.

Parameters **value** (int) – Int value of the color

g

The green component of the color.

Type int

r

The red component of the color.

Type int

set_color_int(*num*)

Assigns the color an integer value representing
the red component bitshifted 24 bits, bitwise ORed with
the green component bitshifted 16 bits, bitwise ORed with
the blue component bitshifted 8 bits, ORed with
the alpha component, or more simply:
r << 24 | g << 16 | b << 8 | a
OR
0xRRGGBBAA

Parameters **num** (int) – Number to set the color to

set_color_rgb(*r=0*, *g=0*, *b=0*, *a=255*)

Assign a value by individual color components.

Parameters

- **r** (int (0-255)) – Red component
- **g** (int (0-255)) – Green component
- **b** (int (0-255)) – Blue component
- **a** (int (0-255)) – Alpha component

to_string_hex()

Returns a hex string representing the color.

Return type class:*str*

nanome.util.complex_save_options module

class MMCIFSaveOptions

Bases: object

Options for saving MMCIF files.
Includes options for writing:
- hydrogens
- only selected atoms

class PDBSaveOptions
Bases: object

Options for saving PDB files.
Includes options for writing:
- hydrogens
- TER records
- bonds
- heterogen bonds
- only selected atoms

class SDFSaveOptions
Bases: object

Options for saving SDF files.
Includes options for writing:
- all bonds
- heterogen bonds

nanome.util.config module

fetch(*key*)

Fetch a configuration entry from your nanome configuration.
Built-in keys are:
host - your NTS server address
port - your NTS server port
key - your NTS key file or string
plugin_files_path - where your plugins will store files

Parameters **key** (str) – The key of the config value to fetch

set(*key*, *value*)

Set a configuration entry in your nanome configuration.
Built-in keys are host, port, key and plugin_files_path.
Default values are 127.0.0.1, 8888, nts_key and ~/Documents/nanome-plugins

Parameters

- **key** (str) – The key of the config value to set
- **value** (str) – The value to set the config item to

nanome.util.enum module

safe_cast
classmethod(function) -> method

Convert a function to be a class method.

A class method receives the class as implicit first argument, just like an instance method receives the instance. To declare a class method, use this idiom:

class C: @classmethod def f(cls, arg1, arg2, ...):

...

It can be called either on the class (e.g. C.f()) or on an instance (e.g. C().f()). The instance is ignored except for its class. If a class method is called for a derived class, the derived class object is passed as the implied first argument.

Class methods are different than C++ or Java static methods. If you want those, see the staticmethod builtin.

nanome.util.enums module

class AtomRenderingMode
Bases: enum.IntEnum

Shape types an atom can be rendered as.
To be used with atom.atom_mode

Adaptive = 6

BFactor = 5

BallStick = 0

Point = 4

Stick = 1

VanDerWaals = 3

Wire = 2

class ColorScheme
Bases: enum.IntEnum

Color schemes for all structure representations.
To be used with plugin_instance.apply_color_scheme

BFactor = 3

Chain = 6

Chothia = 14

DonorAcceptor = 7

Element = 4

Hydrophobicity = 11

IMGT = 12

Kabat = 13

Monochrome = 9

Occupancy = 2

Rainbow = 5

Residue = 1

SecondaryStructure = 8

YRBHydrophobicity = 10

class ColorSchemeTarget

Bases: enum.IntEnum

Structure representations.
To be used with plugin_instance.apply_color_scheme

All = 3

AtomBond = 0

Ribbon = 1

Surface = 2

class ExportFormats

Bases: enum.IntEnum

File export formats.
To be used with plugin_instance.request_export

MMCIF = 3

Nanome = 0

PDB = 1

SDF = 2

SMILES = 4

class HorizAlignOptions

Bases: enum.IntEnum

Horizontal alignment modes for text.
To be used with ui.Label().text_horizontal_align and ui.Button().horizontal_align

Left = 0

Middle = 1

Right = 2

class Integrations

Bases: nanome.util.enums._CommandEnum

An enumeration.

calculate_esp = 2

export_file = 4

export_locations = 5

generate_molecule_image = 6

hydrogens = 0

import_file = 7

minimization = 3

structure_prep = 1

class Kind

Bases: enum.IntEnum

Bond types.
To be used with bond.kind and elements of bond.kinds

Aromatic = 4

CovalentDouble = 2

CovalentSingle = 1

CovalentTriple = 3

Unknown = 0

class LayoutTypes

Bases: enum.IntEnum

Orientation modes for Layout Nodes.
To be used with ui.LayoutNode().layout_orientation

horizontal = 1

vertical = 0

class LoadFileErrorCode

Bases: enum.IntEnum

Errors when loading files into Nanome.
Accessible via the first parameter of the 'done' callback for plugin_instance.send_files_to_load

loading_failed = 1

no_error = 0

class NotificationTypes
Bases: enum.IntEnum

Types of user notifications.
Each value exists as a method on nanome.util.Logs

error = 3

message = 0

success = 1

warning = 2

class PaddingTypes
Bases: enum.IntEnum

UI padding types.
To be used with ui.LayoutNode().padding_type

fixed = 0

ratio = 1

class Permissions
Bases: nanome.util.enums._CommandEnum

An enumeration.

local_files_access = 0

class PluginListButtonType
Bases: enum.IntEnum

Buttons on the plugin list, modifiable by the plugin itself.
To be used with plugin_instance.set_plugin_list_button

advanced_settings = 1

run = 0

class RibbonMode
Bases: enum.IntEnum

Ribbon display modes.
To be used with structure.Residue().ribbon_mode

AdaptiveTube = 1

Coil = 2

SecondaryStructure = 0

class ScalingOptions
Bases: enum.IntEnum

Ways for an image to scale.
To be used with ui.Image().scaling_option

fill = 1

fit = 2

stretch = 0

class SecondaryStructure
Bases: enum.IntEnum

Secondary structure types.
To be used with structure.Residue().secondary_structure

Coil = 1

Helix = 3

Sheet = 2

Unknown = 0

class ShapeAnchorType
Bases: enum.IntEnum

Object type to anchor a Shape to.
To be used with shapes.Shape().anchors

Atom = 2

Complex = 1

Workspace = 0

class ShapeType
Bases: enum.IntEnum

Types of shapes that can be created within Nanome.
Used internally

Label = 2

Line = 1

Sphere = 0

class SizingTypes
Bases: enum.IntEnum

Ways in which a Layout Node can be sized within a UI layout.
To be used with ui.LayoutNode().sizing_type

expand = 0

fixed = 1

ratio = 2

class SkyBoxes
Bases: enum.IntEnum

Preset skyboxes to show in a Nanome room
To be used with plugin_instance.room.set_skybox

Black = 3

BlueSkyAndClouds = 0

BlueSkyAndGround = 2

Graydient = 5

Sunset = 1

Unknown = -1

White = 4

class StreamDataType
Bases: enum.IntEnum

Stream datatypes.
Used internally

byte = 1

float = 0

string = 2

class StreamDirection
Bases: enum.IntEnum

Stream directions (reading and writing).
Used internally

reading = 1

writing = 0

class StreamType

Bases: enum.IntEnum

Object attributes and sets of attributes that can be streamed to Nanome.
To be used with plugin_instance.create_writing_stream and plugin_instance.create_reading_stream

color = 1

complex_position_rotation = 4

label = 3

position = 0

scale = 2

shape_color = 6

shape_position = 5

sphere_shape_radius = 7

class ToolTipPositioning

Bases: enum.IntEnum

Ways in which a tooltip can appear on top of its Layout Node.
To be used with ui.Button().tooltip.positioning_target

bottom = 5

bottom_left = 4

bottom_right = 6

center = 8

left = 3

right = 7

top = 1

top_left = 2

top_right = 0

class VertAlignOptions

Bases: enum.IntEnum

Vertical alignment modes for text.
To be used with ui.Label().text_vertical_align and ui.Button().vertical_align

Bottom = 2

Middle = 1

Top = 0

class VolumeType
Bases: enum.IntEnum

Volume types visible within a complex.
To be used with _internal._volumetric._VolumeData()._type

cryo_em = 3

default = 0

density = 1

density_diff = 2

electrostatic = 4

class VolumeVisualStyle
Bases: enum.IntEnum

Ways that a complex's volume can be displayed.
To be used with _internal._volumetric._VolumeProperties()._style

FlatSurface = 1

Mesh = 0

SmoothSurface = 2

reset_auto()

nanome.util.file module

class DirectoryEntry
Bases: object

Deprecated.

class DirectoryErrorCode
Bases: enum.IntEnum

Deprecated.

folder_unreachable = 1

no_error = 0

class DirectoryRequestOptions
Bases: object

Deprecated.

class DirectoryRequestResult
Bases: object

Deprecated.

class FileData
Bases: object

Deprecated.

class FileError
Bases: enum.IntEnum

File errors encounterable after performing a file operation on the Nanome host machine.
Accessible via the first parameter of the 'done' callback for all methods on plugin_instance.files

invalid_path = 1

io_error = 2

no_error = 0

security_error = 3

unauthorized_access = 4

class FileErrorCode
Bases: enum.IntEnum

Deprecated.

file_unreachable = 1

missing_permission = 3

no_error = 0

path_too_long = 2

class FileMeta
Bases: object

Represents file metadata from a Nanome host machine.
Accessible via the second parameter of the 'done' callback for plugin_instance.files.ls

class FileSaveData
Bases: object

Deprecated.

write_text(*text*)

class LoadInfoDone
Bases: object

Represents the a file operation on the Nanome host machine.
Accessible via the first parameter of the 'done' callback for all methods on plugin_instance.files

ErrorCode
alias of *nanome.util.enums.LoadFileErrorCode*

nanome.util.import_utils module

class ImportUtils
Bases: object

static check_import_exists(*lib_name*)

Used internally.

nanome.util.logs module

class Logs
Bases: object

Allows for easy message logging without buffer issues.
Possible log types are Debug, Warning, and Error.

classmethod debug(**args*)

Prints a debug message
Prints only if plugin started in verbose mode (with -v argument)

Parameters **args** (*Anything printable*) – Variable length argument list

static deprecated(*new_func=None*, *msg=''*)

classmethod error(**args*)

Prints an error

Parameters **args** (*Anything printable*) – Variable length argument list

classmethod message(**args*)

Prints a message

Parameters **args** (*Anything printable*) – Variable length argument list

classmethod warning(**args*)

Prints a warning

Parameters **args** (*Anything printable*) – Variable length argument list

nanome.util.matrix module

class Matrix(*m*, *n*)
Bases: object

Represents a matrix. Used to do calculations within a workspace

classmethod compose_transformation_matrix(*position*, *rotation*, *scale=None*)

classmethod from_quaternion(*quaternion*)

classmethod from_vector3(*vector*)

get_determinant()

get_inverse()

get_minor(*i*, *j*)

get_rank()

get_transpose()

classmethod identity(*size*)

transpose()

exception MatrixException
Bases: Exception

nanome.util.octree module

class Octree(*world_size=5000*, *max_per_node=8*)
Bases: object

Tree containing inserted objects and their positions.
Commonly used to get neighboring objects.

add(*data*, *position*)

Add a data node to the octree.

Parameters

- **data** (object) – Data node to add to the octree
- **position** – Position of this data node

get_near(*pos*, *radius*, *max_result_nb=None*)

Get nodes within the octree neighboring a position.

Parameters

- **pos** (Vector3) – Position to check around
- **radius** (float) – Radius around position where nodes within will be returned
- **max_result_nb** (int) – Maximum number of neighbors to return

get_near_append(*pos*, *radius*, *out_list*, *max_result_nb=None*)

Functions like get_near, but with an externally controlled list.

Parameters

- **pos** (Vector3) – Position to check around
- **radius** (float) – Radius around position where nodes within will be returned
- **out_list** (list) – Parent-scoped list to append search neighbors to
- **max_result_nb** (int) – Maximum number of neighbors to return

move(*data*, *new_position*)

Move a data node in the octree.

Parameters

- **data** (object) – Data node in the octree to move
- **new_position** – New position for the data node

print_out()

Prints out information about the octree.

remove(*data*)

Remove a data node from the Octree.

Parameters **data** (object) – The data to remove from the Octree

nanome.util.process module

class Process(*executable_path=None*, *args=None*, *output_text=None*)
Bases: object

A command-line process wrapper.

args

A list of arguments to pass to the executable.

Type list <str>

cwd_path

The working directory path where the process will be/was executed.

Type str

executable_path

The path to the executable to be run.

Type str

output_text

Whether or not the process will produce text output.

start()

Starts the process.

stop()

Stops the process.

nanome.util.quaternion module

class Quaternion(*x=0*, *y=0*, *z=0*, *w=1*)
Bases: object

A vector that holds 4 values. Used for rotation.

EPS = 1e-06

dot (*other*)

Returns the dot between this and another Quaternion

Parameters **other** (Quaternion) – Quaternion to dot product with

Returns A float value representing the dot product.

Return type float

equals (*other*)

classmethod from_matrix (*matrix*)

Creates a Quaternion from a 4x4 affine transformation matrix.

Parameters **matrix** (list <list <float>>) – A 4x4 affine transformation matrix

Returns A Quaternion representing a rotation.

Return type Quaternion

get_conjugate ()

Returns the conjugate of this Quaternion.

Returns A new Quaternion that is the conjugate of this Quaternion.

Return type Quaternion

get_copy ()

Returns A copy of this Quaternion.

Return type Quaternion

rotate_vector (*point*)

Rotates a vector using this Quaternion.

Parameters **point** (Vector3) – The vector to rotate

Returns A rotated vector.

Return type *vector3*

set (*x*, *y*, *z*, *w*)

w

Returns This quaternion's w component.

Return type float

x

Returns This quaternion's x component.

Return type float

y

Returns This quaternion's y component.

Return type float

z

Returns This quaternion's z component.

Return type float

nanome.util.stream module

class StreamCreationError
Bases: enum.IntEnum

Errors possible during stream creation.

AtomNotFound = 1

NoError = 0

UnsupportedStream = 2

class StreamInterruptReason
Bases: enum.IntEnum

Reasons for stream interruption.

Crashed = 1

StreamNotFound = 0

nanome.util.string_builder module

class StringBuilder
Bases: object

A class to build strings from lists of strings. This class is used internally.

append(*s*)

Converts an object to a string and appends it to this StringBuilder's list of strings.

Parameters **s** – The object to be appended as a string.

append_string(*s*)

Appends a string to this StringBuilder's list of strings.

Parameters **s** – The string to be appended.

clear()

Clears this StringBuilder's list of strings.

to_string(*joiner=''*)

Return a string joined with joiner from this StringBuilder's list of strings.

Parameters **joiner** – The string to join between each element of this StringBuilder's list of strings.

Returns A new string created from this StringBuilder's list of strings.

Return type str

nanome.util.vector3 module

class Vector3(*x=0*, *y=0*, *z=0*)
Bases: object

A vector that holds 3 values. Used for position and scale.

classmethod distance(*v1*, *v2*)

Returns the distance between two vectors.

Parameters

- **v1** (Vector3) – The first vector
- **v2** (Vector3) – The second vector

equals(*other*)

Returns True if the components of this vector are the same as another's.

Parameters **other** (Vector3) – The other Vector3

Returns Whether or not this vector is component-equal to 'other'

Return type bool

get_copy()

Returns A copy of this vector.

Return type Vector3

set(*x*, *y*, *z*)

Parameters

- **x** (float) – The x component to set this vector to
- **y** (float) – The y component to set this vector to

- **z** (float) – The z component to set this vector to

unpack()

Returns a 3-tuple containing this vector's x, y, and z components.

Return type tuple

x

The x component of this vector

Type float

y

The y component of this vector

Type float

z

The z component of this vector

Type float

Submodules

nanome.plugin_init module

main()

nanome.setup_config module

display_help()

interactive_mode()

main()

parse_args()

parse_value(*str*, *parser*)

Overview

The Nanome Plugin API provides a way to interface and integrate external software with Nanome's molecular modeling VR software. Through this API, users can link up external computational such as molecular dynamics, docking software, and link custom databases. The extended functionality includes the ability to create new windows inside of the virtual environment and is easily customizable through a drag and drop user interface.

Plugins can be designed and ran from different operating systems - Windows, Linux, and Mac depending on the requirements needed from each plugin.

Some examples of plugins that our customers love are:

- Docking

- Chemical Interactions
- Electrostatic Potential Map generation
- Chemical Properties
- Custom Database Integrations
- Loading PDFs and PowerPoints
- Running custom molecular dynamics
- All of our public plugins are available on our *Github <https://github.com/nanome-ai>* (prefixed with "plugin-").

The primary requirements for running plugins are the Nanome Virtual Reality Software and access to the Nanome Plugin Server (NTS). The Nanome Plugin Server acts as a relay to forward plugin information and processes data coming into and going out of the Nanome virtual environment.

The Nanome Virtual Reality Software can be acquired directly from Nanome or in any of the VR stores here:

- Oculus Store: https://www.oculus.com/experiences/rift/1873145426039242
- Viveport: https://www.viveport.com/apps/0a467f78-2ed2-43eb-ada8-9d677d5acf54
- Steam: https://store.steampowered.com/app/493430/Nanome/
- Direct Download: https://nanome.ai/setup
- SideQuest: https://xpan.cc/a-333

1.8 Using Plugins

Please contact sales@nanome.ai to enable your account to use Plugins.

In order to use a plugin

Make sure you are fully connected to your Nanome plugin server. After the Nanome team has configured your account to use Plugins and has provided the target server location and port (NTS DNS and Port, e.g. organization.nanome.ai 8888), log into Nanome, create a room, and click on the purple "Stacks" button to the left of the Entry list. You should see an empty list or a list of plugins. If you see "Not connected to NTS", please contact support@nanome.ai or your dedicated Account Manager.

Editing the Config File

First, you want to locate the Config file (nanome-config.ini) of the Nanome Application in the builds folder. If you downloaded Nanome through the Oculus store, it will be available here:

C:\Program Files\Oculus\Software\Software\nanome-nanome\Build

Open the nanome-config.ini file in a text editor and scroll down to the section named ' Nanome plugin server config' and change to the following:

Plugin-server-addr = 127.0.0.1

Plugin-server-port = 8888

Now, we want to check to make sure that the Plugin Server is connected. Go ahead and launch Nanome, then log in using your credentials. Create a room and Start in 2D and click on the Plugins Icon on the bottom of the Entry Menu.

You should see that the NTS is connected and there are no current running plugins. If it says that "No NTS is connected", that means it is unable to see the Plugin server and it is entered incorrectly on the Config file or in the Admin settings for home.nanome.ai. It could also be blocked by firewall.

Let's go ahead and run a basic plugin to make sure it is working.

Installing your first plugin: Basic Plugin

Example Plugin

First, download the RemoveHydrogen.py basic plugin here:

This is a simple plugin example to remove all of the selected hydrogens in the workspace:

```
import nanome
from nanome.util import Logs

# Config

NAME = "Remove Hydrogens"
DESCRIPTION = "Remove hydrogens in all selected atoms"
CATEGORY = "Simple Actions"
HAS_ADVANCED_OPTIONS = False

# Plugin

class RemoveHydrogens(nanome.PluginInstance):
    # When user clicks on "Activate"
    def start(self):
        Logs.message("Connected to a new session!") # Displays a message in the␣
↪console

    @staticmethod
    def _should_be_removed(atom):
        if atom.selected == False:
            return False
        if atom.symbol != 'H':
            return False
        return True

    # When user clicks on "Run"
    def on_run(self):
        self.request_workspace(self.on_workspace_received) # Request the entire␣
↪workspace, in "deep" mode

    # When we receive the entire workspace from Nanome
    def on_workspace_received(self, workspace):
        for complex in workspace.complexes:
            count = 0
            for residue in complex.residues:

                # First, find all atoms to remove
                atoms_to_remove = []
                for atom in residue.atoms:
                    # If this atom is an H and is selected, delete it
                    if RemoveHydrogens._should_be_removed(atom):
                        atoms_to_remove.append(atom)

                # Then, remove these atoms
                for atom in atoms_to_remove:
                    residue.remove_atom(atom)
                count += len(atoms_to_remove)
```

(continues on next page)

(continued from previous page)

```
            Logs.debug(count, "hydrogens removed from", complex.molecular.name) #␣
↪Displays a message in the console only if plugin started in verbose mode

        self.update_workspace(workspace) # Update Nanome workspace, in "deep" mode

# Setup plugin information, register RemoveHydrogens as the class to instantiate, and␣
↪connect to the server
nanome.Plugin.setup(NAME, DESCRIPTION, CATEGORY, HAS_ADVANCED_OPTIONS,␣
↪RemoveHydrogens)
```

1.8.1 Development

In order to prepare your development environment and create your own first plugins, follow this link:

Installation

In order to install the Nanome Plugin API, you need a supported version of Python. Then, use python's package manager, pip, to install nanome:

```
$ pip install nanome
```

Or, to upgrade your current installation:

```
$ pip install nanome --upgrade
```

Server

A Nanome Transport Server (NTS) is required to run your plugins and connect them to Nanome. A public server will be available in the near future. If you need a NTS, please contact us.

Running Your First Plugin

Starting a plugin is fairly easy. If you copy-pasted the example plugin on the home page, in a file named "RemoveHydrogens.py", you can start your plugin like this:

```
$ python RemoveHydrogens.py
```

Or on Linux (python 3 is still preferred when available):

```
$ python3 RemoveHydrogens.py
```

To choose the IP address and the port of your server, you have two options:

Short term, testing: **Using arguments**

```
$ python RemoveHydrogens.py -a 123.456.789.0 -p 4567
```

Long term, for production: **Changing the script** (call to plugin.run, last line of the example script above)

```
plugin.run('123.456.789.0', 4567)
```

Arguments

When starting a plugin, a few optional arguments are available:

- -h: Displays available arguments
- -a [IP]: Specifies NTS address
- -p [PORT]: Specifies NTS port
- -k [FILE]: Specifies a key file to use (if NTS is protected by key)
- -v: Enables verbose mode, to display *debug()* messages
- -r: Enables Live Reload

On the VR Side

In order to connect Nanome (VR) to your server, make sure that its configuration file (nanome-config.ini, located in its installation directory) contains the following:

```
plugin-server-addr                          = 127.0.0.1     # Use the␣
↪correct address for your server
plugin-server-port                          = 8888          # Use the␣
↪correct port for your server
```

Our Plugins

We have a growing list of plugins available on our Github (all repositories starting with "plugin-")

In order to install them, you have 2 possibilities: Use pip or manually download them from github.

Using pip

This is the easiest way. For instance, to install and run URLLoader, simply use:

```
$ pip install nanome-loaders
$ nanome-url-loader -a address_of_your_nts
```

And it will be up and running Please refer to each individual repository README for more information about our plugins

Python Module Index

n

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z