Megascan Link Painter

Luca Faggion

Feb 11, 2022

CONTENTS:

CHAPTER

ONE

USER GUIDES

1.1 How to Install

1.1.1 Download the plugin

You can download the plugin in the realease tab of the github project page

Build types

- Development Build

  This build is always updated as soon a commit is pushed on the master branch

  Warning: This builds can be very unstable or not working at all! So when use them expect them to not work or not behaving correctly

- Tagged Builds

  The tagged builds are stable usable builds

1.1.2 Install the Plugin

The plugin is divided in two separated packages that work togheter to import the resources from [Quixel Bridge][quixelbridge] to [Substance Painter][sbspainter]

To install them unzip the archive in the Substance Painter Document directory that is located in:

Note: **For Windows 10** `%userprofile%\Documents\Allegorithmic\Substance Painter` or `%userprofile%\OneDrive\Documents\Allegorithmic\Substance Painter`

Note: **For Linux** `~/Documents/Allegorithmic/Substance Painter`

Note: **For MacOS** `/Users/%username%/Documents/Allegorithmic/Substance Painter`

If you want to install it by hand simply place respectively:

- the `megascan_link_python` folder to `%substance painter documents path%\python\plugins\` folder
- the `megascan_link_js` folder to `%substance painter documents path%\plugins\` folder

Note: if you had Substance Painter opened during the installation you should not be able to see the plugins listed to fix it simply click `Reload Plugin Folder` for both the Python plugin and the Javascript plugin

---

Warning: The plugins packages work together so both plugins should be enabled on Substance Painter!

1.2 How to Use

1.2.1 Basic Usage

In this gif is shown how to import Megascan Assets in an already opened project

---

Note:

Be sure to change the NORMAL MAP to the Correct COLOR SPACE which is OPENGL

1.2.2 Features

Create new project

You can create a new project by exporting from Quixel bridge an asset that contains a 3D Mesh.

Select meshes

When creating a new project if you exported from Quixel bridge multiple assets contianing a 3D Mesh you can select which one use as a base mesh inside Substance Painter, the rest of the assets is imported as resources.

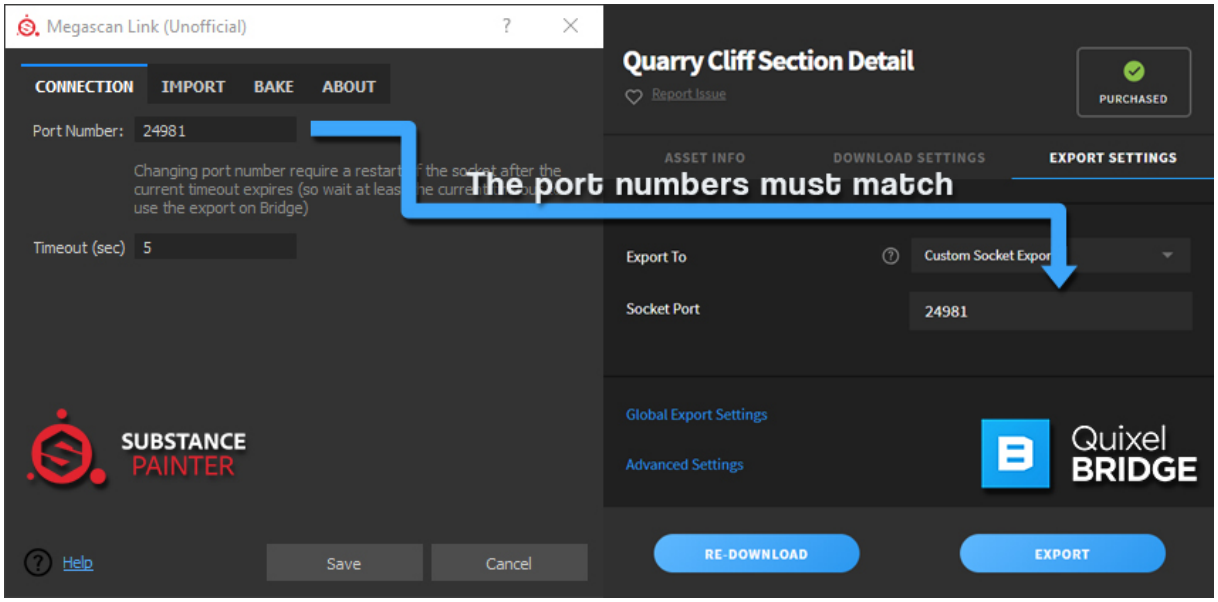
1.2.3 Plugin Settings

Here is the list of settings the plugin exposes

Connection options

This options are used for the connection with Quixel Bridge make sure that the **Port number** is equal to the Port number in the **Custom Socket Export** option in **Quixel Bridge**

Warning: Be sure to insert the same port you have set in the **Export** options of Quixel Bridge



Import options

This options are applied to every import of Megascan Assets

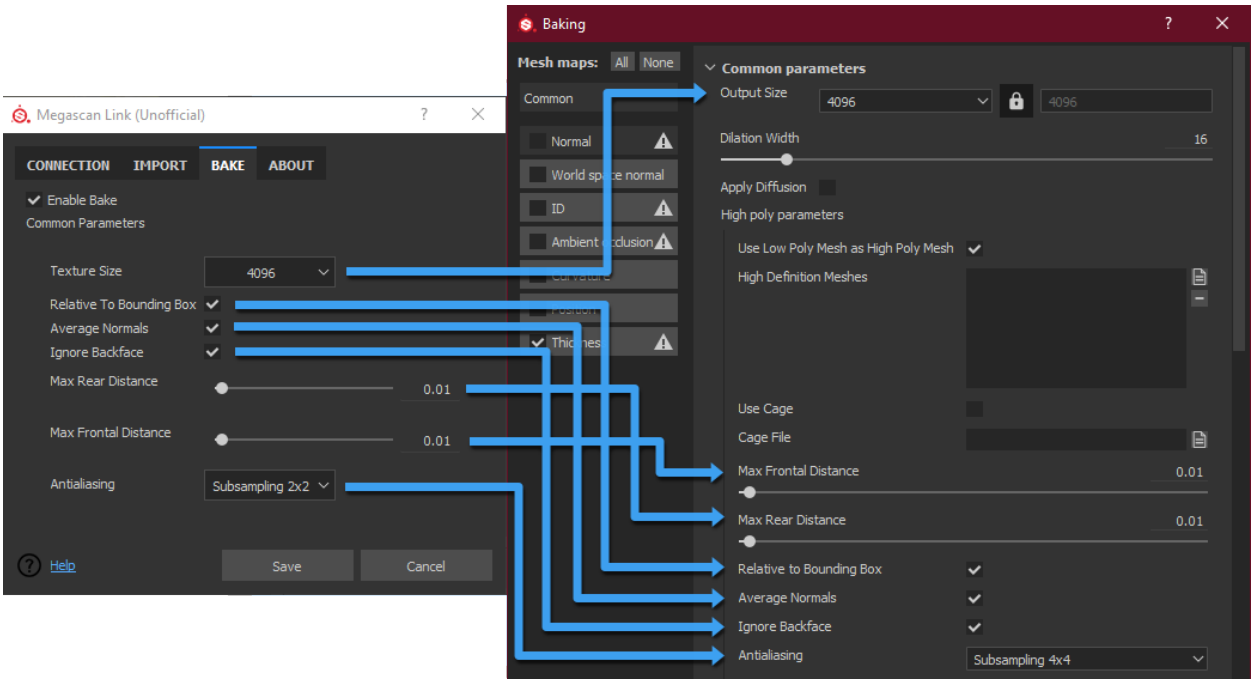
Bake option

Enabling this option allow the plugin to perform a bake, with the preset values, after a successful import and project creation.

The bake is performed only when creating a new project (**New Project** pressed during the import, if there is an already project open, or when importing a Megascan Asset containing a 3D Mesh when there is no project open).

If the imported Megascan Asset has a High poly mesh the **High Poly (HP)** mesh list is automatically populated with the correct meshes.

Note: The options are the same as the ones in the bake window of Substance Painter


Megascan Link (Unofficial)
CONNECTION
IMPORT
BAKE
ABOUT
Enable Bake
Common Parameters
Texture Size
4096
Relative To Bounding Box
Average Normals
Ignore Backface
Max Rear Distance
0.01
Max Frontal Distance
0.01
Antialiasing
Subsampling 2x2
Help
Save
Cancel
Baking
Mesh maps:
All
None
Common
Normal
World space normal
ID
Ambient occlusion
Thickness
Common parameters
Output Size
4096
4096
Dilation Width
16
Apply Diffusion
High poly parameters
Use Low Poly Mesh as High Poly Mesh
High Definition Meshes
Use Cage
Cage File
Max Frontal Distance
0.01
Max Rear Distance
0.01
Relative to Bounding Box
Average Normals
Ignore Backface
Antialiasing
Subsampling 4x4

CHAPTER

TWO

DEVELOPER DOCS

2.1 Megascan Link Python Package

This package is the connection between Quixel Bridge and Substance Painter

The role of this package is to receive the JSON data from Bridge and send it over websocket to the JS counterpart that will use that data to import the needed resources and to store and manage the configuration settings (*Config Module*).

The plugin has been divided into two plugins *Megascan Link Python Package* and *Megascan Link Javascript Package* because of the current state of the Substance Painter API, what is possible with the JS API in not possible with the Python API and the other way around. Since this plugin makes use of some features that are not present in both API I had to split the plugin in half.

2.1.1 Python Subpackages

Ui package

UI Submodules

Ui Module contents

This package contains all the custom controls and the ui design files used to create all the dialogs adn windows used by the Python plugin

Icon Module

Module containing classes for using and retriving icons files

megascan_link_python.ui.icon.**getIcon**(*name: str*) → str

Return the path to the specified icon :param name: icon filename :type name: str :return: absolute path to the icon :rtype: str

megascan_link_python.ui.icon.**getIconAsQPixmap**(*name: str*, *scale: int = None*) → PySide2.QtGui.QPixmap

Retrive a icon image as a QPixmap

Parameters

- **name** (*str*) – the icon filename
- **scale** (*int, optional*) – the desired size to apply to the icon (squared size), defaults to None

Returns the icon as a QPixmap

Return type QtGui.QPixmap

2.1.2 Python Submodules

2.1.3 Plugin Module contents

class megascan_link_python.**Data**
Bases: object

Dataclass used to store references to items so they dont get garbage collected

socket = None

toolbar = None

megascan_link_python.**close_plugin**()
Exit point of the plugin.

Here we perform the clean up before closing the plugin, stopping the socket thread and removing the toolbar action

megascan_link_python.**createToolBar**()
Creates the toolbar containing the action to open the Settings Dialog

megascan_link_python.**openSettingsDialog**()
Opens the Setings dialog for the user to change the socket port number and other import settings

megascan_link_python.**showErrorDialog**()
Opens the error dialog showing to the user that something went wrong during the installation of the needed dependencies.

And guiding the user to the manual dependecies installation guide on the documentation website

megascan_link_python.**start_plugin**()
Entry point of the plugin.

Here we set up all the needed fucntionalities like the log the socket thread and we add to the painter toolbar the user interface ation

2.1.4 Config Module

Module containing classes for managing the config settings files or related

class megascan_link_python.config.**ConfigSettings**
Bases: object

Class that manages a config file

classmethod checkConfigState()
Check if the current config file is opened if not and the file exist reads and load the content of it to the config parser

classmethod checkIfOptionIsSet(*cat: str*, *prop: str*, *fallback=''*) → bool
Helper function that will check if a propriety of a section is set or not by confronting it with the following values [“true”, “yes”, “y”, “ok”]

Parameters

- **cat** (*str*) – Category name string

- **prop** (*str*) – Propriety of the category to check agains

Returns if the propriety is set returns True, False otherwise

Return type bool

config = <configparser.ConfigParser object>
Config parser class instance

classmethod flush()
Helper function used to write the content to file

classmethod getAsDict() → dict
Helper function that return the current config as a Python dictionary

Returns a dictionary of the current ini file

Return type dict

classmethod getConfigCategory(*cat: str*) → dict
Helper function to retrive an entire category of the ini file

Parameters **cat** (*str*) – Category name string

Returns the dictionary containing the proprities with their corrispective values, if the category does not exist and empty dictionary is returned

Return type str

classmethod getConfigSetting(*cat: str*, *prop: str*, *fallback=''*) → str
Helper function to retrive a config propriety value.

Parameters

- **cat** (*str*) – Category name string
- **prop** (*str*) – Propriety of the category to retrive

Returns the propriety value

Return type str

classmethod getConfigSettingAsList(*cat: str*, *prop: str*, *separator=','*) → List[str]
Helper function to retrive a config option as a list

Parameters

- **cat** (*str*) – Category name string
- **prop** (*str*) – Propriety of the category to retrive as list
- **separator** (*str, optional*) – the saparator character used to split values in the config string value, defaults to ","

Returns the config settings as a List

Return type List[str]

opened = False
Current state of the config file

path = None
Contains the path to the ini config file (root dir of module) needs to be initialized

classmethod removeConfigSettings(*cat: str*, *prop: str*, *flush=True*) → bool
Helper function used to clear an option entry of a Section

Parameters

- **cat** (*str*) – Category name string
- **prop** (*str*) – Propriety of the category to remove
- **flush** (*bool, optional*) – If true it will immediatly update the file on disk, defaults to True

Returns return True if successfully delete False otherwise

Return type bool

classmethod setIniFilePath(*name: str*)
Set the filepath of the ini file

Parameters **name** (*str*) – the name of the config ini file (without extension)

classmethod setUpInitialConfig(*config: configparser.ConfigParser*)
Function to use a config parser instance to initialize the config file This will initialize the config file only if it does not exist

Parameters **config** (*configparser.ConfigParser*) – The config instance to use for populating the initial value of the config

classmethod updateConfigSetting(*cat: str*, *prop: str*, *value: str*, *flush=True*)
Helper function used to update a config propriety.

Parameters

- **cat** (*str*) – Category name string
- **prop** (*str*) – Propriety of the category to update
- **value** (*str*) – Value to associate to the propriety
- **flush** (*bool, optional*) – If true it will immediatly update the file on disk, defaults to True

2.1.5 Dialogs Module

Module which contains all the dialogs used by the plugin

The dialogs are generated using QtDesigner and are located under /ui/uiDesign and then converted to python code using the buildDialogs.py script

class megascan_link_python.dialogs.**DependencyErrorDialog**(*parent*, *helpLink=None*)
Bases: PySide2.QtWidgets.QDialog, megascan_link_python.ui.error_dialog.Ui_Dialog

Generic Error dialog for displaying error messages

close()
Close the dialog and updates the ini file if necessary

openHelp()
Summon a browser with the documentation page opened

show()
Shows the error dialog only if the users has not checked before the "don't show again" checkbox

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class megascan_link_python.dialogs.**SettingsDialog**(*socket: megascan_link_python.sockets.SocketThread*, *parent=None*)

Bases: PySide2.QtWidgets.QDialog, megascan_link_python.ui.settings_dialog.Ui_Dialog

Dialog displayed to the user for editing the plugin settings

staticMetaObject = <PySide2.QtCore.QMetaObject object>

2.1.6 Log Module

This Module contains the logger facilities class

class megascan_link_python.log.**LoggerLink**

Bases: object

Class used to log messages to the log file

see: Log() for know how to use it to print also to the Python editor output

classmethod Log(*msg: str*, *logLevel=20*)

Helper function used to log a massage to a file or if specified in the config file with the *outputConsole* propriety also to the Python Editor output of Substance Designer

Parameters

- **msg** (*str*) – the message to print
- **logLevel** (*int, optional*) – the log level to print with if it is lower than the current _logger level it would not be printed, defaults to logging.INFO

classmethod setLoggerName(*name: str*)

Set the current session logger name do this before using the logger and once, subseguent calls to this methos will have no effect

Parameters **name** (*str*) – the logger name

classmethod setUpLogger()

Method used to setup the current logger instance

Links the handler to print to the log file (log config path: './referencefixer.log') and set up the format to print with

2.1.7 Sockets Module

Module containing classes for managing the comunincation with the socket thread and the main thread (in which we can use the SDAPI)

class megascan_link_python.sockets.**SocketThread**

Bases: PySide2.QtCore.QThread

Core plugin class that manages a socket process for receiving TCP packets from Quixel Bridge

close()

Set the needed flags to close the socket

Note: The close operation is performed only after the timeout duration

onDataReceived = <PySide2.QtCore.Signal object>
Signal that is fired whenever a packet is retrived over the socket

Type QtCore.Signal

Parameters **object** – json dictionary containing the data

restart()
Set the needed flags to perform a socket restart

Note: The restart is performed only after the timeout duration

run()
This is the method that manages the socket lifetime process

To interact with the socket use the *close()* and *restart()* method instead

While this method is running the associated thread is kept alive **closing** the socket without requesting a **restart** will make this thread close too

The socket is listening on the port specified on the config file and it is restarted every time the timeout duration expires (also setted from the config file)

shouldClose = False
flag that indicates that the socket should stop in the next timeout frame

see *close()*

Warning: dont use this flag directly use instead the *close()* methods

shouldRestart = False
flag that indicates that a restart is been requested, the restart is processed in the next timeout frame, it is cleared (False) when the restart happen

used for example if you want to change the listening port or the timeout duration

see *restart()*

Warning: dont use this flag directly use instead the *restart()* method instead

started = False
variables that idicates if the socket has been started

but it is not guarantee that it is listening

staticMetaObject = <PySide2.QtCore.QMetaObject object>

2.1.8 Utilities Module

Contain Helper Functions used in common tasks

megascan_link_python.utilities.**getAbsCurrentPath**(*append: str*) → str
Simple function to get the current script path

Parameters **append** (*str*) – path or filename to add

Returns the full path plus the append param

Return type str

2.1.9 Websocket Link Module

Module containing classes used to send data over the JS Plugin

class megascan_link_python.websocket_link.**WebsocketLink**(*parent=None*)
Bases: PySide2.QtCore.QObject

Start up a single use websocket to send data over the JS plugin

sendDataToJs(*data: object*)
send the data to the JS plugin using a websocket with port 1212

Parameters **data** (*object*) – the json data to send

staticMetaObject = <PySide2.QtCore.QMetaObject object>

2.2 Megascan Link Javascript Package

This package is responsible to utilize the Substance Painter API to import the Megascan Assets into the current opened project.

It receives the data over websocket from the *Megascan Link Python Package*.

2.2.1 Utility

checkIfSettingsIsSet(*setting*)
Helper function that will check if a propriety of a section is set or not by confronting it with the following values [“true”, “yes”, “y”, “ok”]

Arguments

- **setting** (*String*) – String value to check

getHpMeshes(*asset*)
Extract for the lod list of an asset the High Poly meshes

Arguments

- **asset** (*Object*) – the asset from which extract the High Poly meshes

Returns **Array** – the list of High poly meshes paths

removeFromAssets(*asset*, *assets*)
Filter the Megascan Asset list removing a specific asset

Arguments

- **asset** (*Object*) – the asset to remove
- **assets** (*Array*) – the list of asset to filter

2.2.2 String Extensions

`String.`**format**
Simple extension to the String object that allow to format a string python style taken from StackOverflow

2.2.3 QML Modules

class main : **public** PainterPlugin

Public Functions

void **importResources** (Object *data*)
Import the Megascan Assets textures in the project and put them in the Megascan/{AssetName} path for each asset
Parameters
- `data`: Quixel Bridge Json data

void **setUpAndBake** (Object *asset*)
Set up the baking parameters based on the user config preset parameters and then perform the bake
Parameters
- `asset`: the asset to retrive the bake data from

void **createProjectWithResources** (Object *asset*, Array *imports*)
Create a new project using the defined asset as the Mesh Asset
Parameters
- `asset`: Json data of the Mesh asset (Quixel Bridge)
- `imports`: Array of Json data of additonal resources to import in the newly created project

boolean **checkForMeshAssets** (Object *data*)
Check if in the Bridge json data are present some Assets that have associated a Mesh Asset
Return returns True if there is an 3D Asset, False otherwise
Parameters
- `data`: Quixel Bridge Json data to check

Public Members

bool projectCreated READ dummyGetter_projectCreated_ignore
global flag that indicates that a project has been created

Object settings READ dummyGetter_settings_ignore
the settings values of the user config file

Object meshAsset READ dummyGetter_meshAsset_ignore
the current Megascan 3D asset used in the project

Private Members

var Component onCompleted
Simply indicates that the plugin has been loaded correctly

var **onActiveTextureSetChanged**
Used to set up the textures set resolution and channel if a new project is being create by this plugin Also set up and started, if requested in the config file, the baking process

AlgDialog **saveError**
Dialog showed to the user when the plugin can't save the current opened project

This happens for example when the users as project opened that has not been saved to disk

AlgSelectDialog **selectMesh**
Instance of *AlgSelectDialog*, this dialog is used for starting a new project with the mesh selected by the user from a list of meshes in the megascan data currently being imported, if the are none or only one 3D mesh this dialog is not showed

AlgNewProject **createProject**
Instance of *AlgNewProject*, this dialog is showed to the user if there is a 3D Mesh in the currently imported data.

As Substance Painter only supports a single mesh per project we ask the user if he want to create a new project with the the data he is currently importing.

This dialog can be suppressed with the config option `askcreateproject` to `False`

class AlgNewProject : **public** AlgDialog
This dialog is presented to the user when there is a 3D Mesh asset in the import data and the config option `askcreateproject` is set to `True`

The dialog asks the user if he want to import the assets bitmaps (since Painter only supports a single mesh per project) or wants to create a new project with one of them as the base mesh.

Public Functions

void **openWithData**(Object *data*)
Open the dialog feeding the JSON data coming from Quixel Bridge
Parameters
- `data`: the Quixel Bridge JSON data

Public Members

Object importData READ dummyGetter_importData_ignore
the JSON data currently being imported

Signals

void **importPressed**(var *importData*)
Emitted when the user press the **Import** button
Parameters
- `importData`: the JSON data currently being imported

void **newProjectPressed**(var *importData*)
Emitted when the user press the **New Project** button
Parameters
- `importData`: the JSON data currently being imported

Private Members

var **title**
The tile of the dialog

var **visible**
Visibility of the dialog - default is false

var **width**
width of the dialog -size if fixed to `400x150`

var **height**
height of the dialog -size if fixed to `400x150`

var **maximumHeight**
is set to be equal to the `height` variable this blocks the user to resize the dialog

var **maximumWidth**
is set to be equal to the `width` variable this blocks the user to resize the dialog

var **minimumHeight**
is set to be equal to the `height` variable this blocks the user to resize the dialog

var **minimumWidth**
is set to be equal to the `width` variable this blocks the user to resize the dialog

AlgButton **importBtn**
Dialog Import button - emits the **importPressed** signal

AlgButton **newPrjBtn**
Dialog New Projecy button - emits the **newProjectPressed** signal

namespace QtQuick

namespace Painter

namespace QtWebSockets

namespace AlgWidgets

class AlgSelectDialog : **public** AlgDialog
This dialog is presented to the user if, when creating a new project, there is more than one 3D Mesh asset in the import data.

This dialogs allow the user to select a 3D Asset to use a the base mesh for the new project, if there are other 3D Assets only their bitmaps are imported.

Public Functions

void **addAssets**(assetsIn)
Add assets to the list widgets of the dialog
Parameters
- `the`: list of assets to add

void **openWithAssets**(assetsIn)
Shorthand function to open the dialog and populate the asset list
Parameters
- `the`: list of assets to add to the dialog list

Public Members

Object assets READ dummyGetter_assets_ignore
the list of 3d assets

Object importData READ dummyGetter_importData_ignore
the list of 3d assets

int currentIndex READ dummyGetter_currentIndex_ignore
the current index selected in the list

Private Members

var **title**
The tile of the dialog

var **width**
width of the dialog -size if fixed to `400x300`

var **height**
height of the dialog -size if fixed to `400x300`

var **maximumHeight**
is set to be equal to the `height` variable this blocks the user to resize the dialog

var **maximumWidth**
is set to be equal to the `width` variable this blocks the user to resize the dialog

var **minimumHeight**
is set to be equal to the `height` variable this blocks the user to resize the dialog

var **minimumWidth**
is set to be equal to the `width` variable this blocks the user to resize the dialog

var **defaultButtonText**
the accept button text of the dialog - default is `Select`

ListView **assetListView**
The List widget (ListView QML)

ListModel **assetList**
The model of the list, it holds the data needed to populate the widget

Component **headerList**
The header of the list

Component **assetListDelegate**
This is component instanciated for each entry in the model (*assetList*)

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

L

M

O

P

Q

R

S

T

U

W