Megascan Link

Luca Faggion

Jun 10, 2020

CONTENTS:

CHAPTER

ONE

PLUGIN GUIDE

1.1 How to use the plugin

1.1.1 Download the plugin

You can find it in the realease tab of the github project

Build types

- Development Buid

  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 Buils

  The tagged builds are stable usable builds

1.1.2 Install the plugin

Install it in Substance Designer using the Plugin manager, you can find it under `Tools > Plugin Manager..` then click `install` and navigate to the path were you previously downloaded it

Note: Refere to the official doc for installing a Plugin Package

1.1.3 Use the Plugin

After you have installed the plugin go to **Quixel Bridge** select the **Megascan asset** you want to import on Substance Designer go to the **Export Setting** tab and select from the **Export To** drop down the **Custom Socket Export** option, then in the **Socket Port** insert the same port you have set up on the **Plugin Settings** (Default to **24891**)

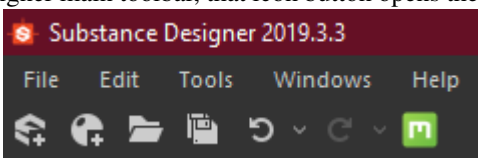
Bridge setup example

Substance designer import step

When presented the import dialog you can choose to wich packages import the Megascan assets by selecting them in the list (shift selection and ctrl selection are enabled) and then click import

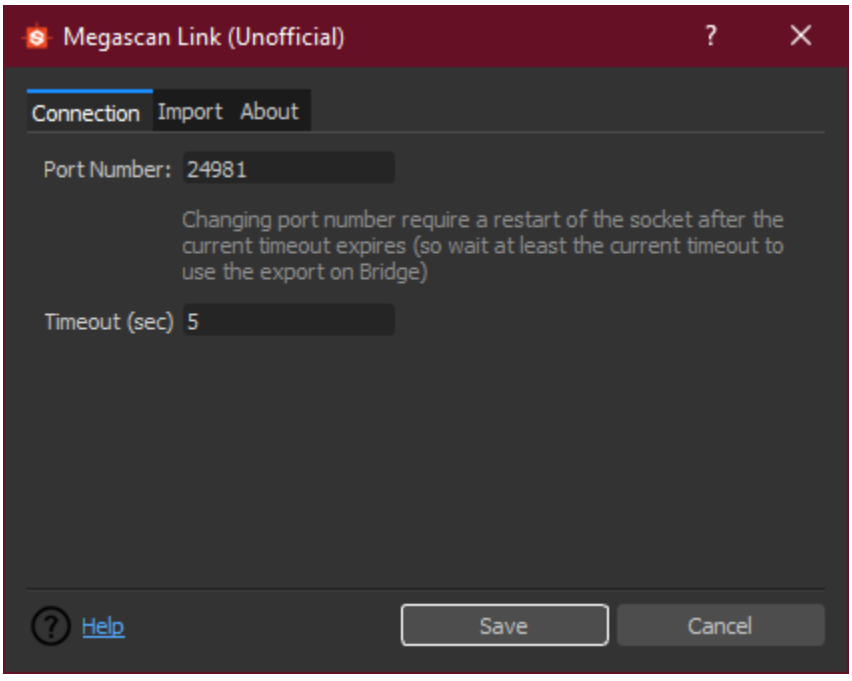
1.1.4 Plugin Settings

When you installed the plugin a new icon appeared in the Substance Designer main toolbar, that icon button opens the Plugin Settings dialog that can be used to change how the plugin works.



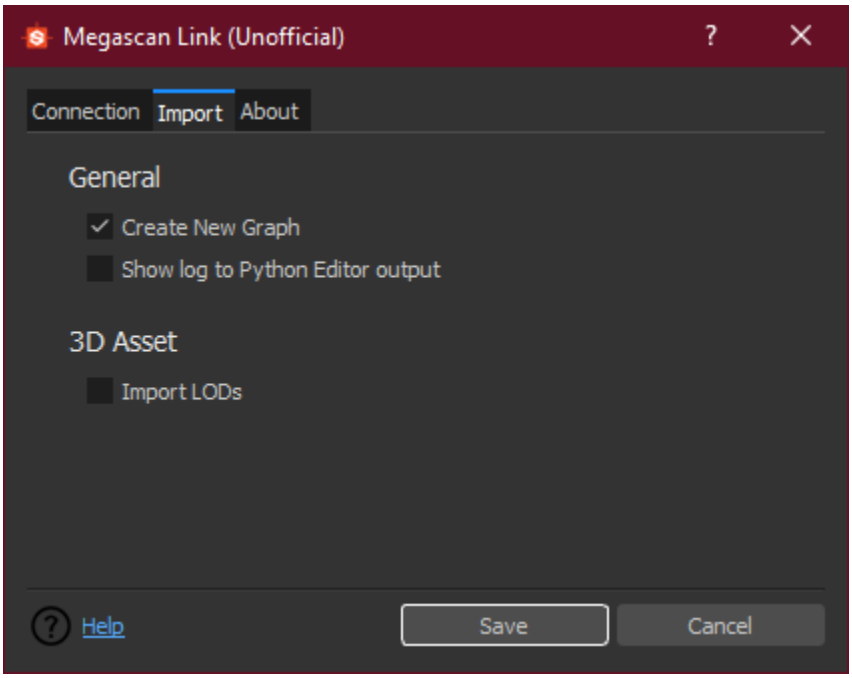
Connection settings

This tab allow to change the connection setting of the socket



Import settings

This tab allow to change the import behaviour of the plugin



CHAPTER

TWO

DEVELOPER DOCS

2.1 Development Prerequisites

There are no prerequisites for the plugin all the requisites are already build-in in the default Python installation of Substance Designer

The only requisite necessary is PySide2 for using the `buildDialogs.py` script.

But if you want to debug the plugin with a step-through debugger refere to the official Substance Designer documentation. Debugging Plugins using Visual Studio Code

2.1.1 Building the Documentation

The prerequisite to build the documentation using **sphinx** are the following:

when all the prerequisites are installed these are the steps needed to build the documentation:

- Navigate to the doc folder

```
cd doc
```

- Build using make

```
make html
```

2.1.2 Building the sdplugin package

To build the sdplugin package navigate to the root folder of the repository and simply execute the `makepackage.py` script:

```
python makepackage.py
```

the resulting builded file is placed in the `build` directory

2.2 Megascan Link package

2.2.1 Subpackages

Ui package

Submodules

Import_dialog module

class megascan_link.ui.import_dialog.**Ui_Dialog**
Bases: object

retranslateUi(*Dialog*)

setupUi(*Dialog*)

Settings_dialog module

class megascan_link.ui.settings_dialog.**Ui_Dialog**
Bases: object

retranslateUi(*Dialog*)

setupUi(*Dialog*)

Module contents

2.2.2 Submodules

2.2.3 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.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>

2.2.4 ResourceImporter module

Contains classes to import the data from Quixel Bridge to Substance Designer

class megascan_link.resourceImporter.**BitmapType**
Bases: enum.Enum

Enum class used to have a quick access to the corrensponding SDUsage Since the data from Quixel Bridge comes as a string we can get the corresponding SDUsage simply using BitmapType[str]

class megascan_link.resourceImporter.**MegascanBitmap**(*resource: sd.api.sdresourcebitmap.SDResourceBitmap*, *path: str*, *usage: str*, *name=None*)
Bases: object

Wrapper class composed of the data coming from Quixel Bridge and the corrispective SDResourceBitmap

getUsageArray() → sd.api.sdvaluearray.SDValueArray
Method returning an SDValueArray with the usage of the texture

Returns SDValueArray with the usage of the texture

Return type SDValueArray

class megascan_link.resourceImporter.**ResourceImporter**(*parent=None*)
Bases: PySide2.QtCore.QObject

Class responsible of importing to Substance Designer all the meshes/bitmaps/data contained in the payload from Quixel Bridge

createGraphWith(*graphname: str*, *bitmaps: List[*megascan_link.resourceImporter.MegascanBitmap*]*, *package: sd.api.sdpackage.SDPackage*)
Create a graph in the specified package using the specified bitmaps creating their respective outputs and finalally linking them as a bonus we set the graph icon to the Megascan Logo :)

Parameters

- **graphname** (*str*) – name of the new graph
- **bitmaps** (*List[MegascanBitmap]*) – List of MegascanBitmap to use in the graph
- **package** (*SDPackage*) – the package reference where to create the graph

data = None
Current data payload being processed

importFromData(*data*)
Entry point for the data coming from the socked thread

Parameters **data** (*List[dict]*) – Json Quixel Bridge data

processImportForPacakges(*packages*)
this is the actual method that performs the bitmaps/meshes import and graph creation tasks it does it for every package that the user selected this method uses the class attribute data and clears it when done and look up for the settings from the configuration file

Parameters **packages** (*List[SDPackage]*) – List of SDPackage where to import the currently processing data

staticMetaObject = <PySide2.QtCore.QMetaObject object>

2.2.5 Config module

Module containing classes for managing the config settings files or related

class megascan_link.config.**ConfigSettings**
Class that manages a config file

path = './megascanlink.ini'
Contains the path to the megascanlink.ini config file (root dir of module)

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*) → 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 getConfigSetting(*cat: str*, *prop: str*) → 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

opened = False
Current state of the config file

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

2.2.6 Dialogs module

Module which contains all the dialogs used by the plugin

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

class megascan_link.dialogs.**SelectPackageDialog**(*packageList*, *parent=None*)
Bases: PySide2.QtWidgets.QDialog, *megascan_link.ui.import_dialog.Ui_Dialog*

Dialogs displayed when an import is requested from Quixel Bridge

The user can select to which packages import the Megascan Assets or dismiss it

returnValue = <PySide2.QtCore.Signal object>
Subscribable signal emitted when the user close the dialog the param is set to the list of the selected packages

Type QtCore.Signal

selectedPackages = None
List of currently selected packages the data value is set to point to the corresponding SDPackage Reference

Type List[QtWidgets.QListWidgetItem]

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class megascan_link.dialogs.**SettingsDialog**(*socketRef:* megascan_link.sockets.SocketThread, *parent=None*)
Bases: PySide2.QtWidgets.QDialog, *megascan_link.ui.settings_dialog.Ui_Dialog*

Dialog displayed to the user for editing the plugin settings

saveSettings()
Saved the changed settings to file then inform the socket thread if it need to restart himself

staticMetaObject = <PySide2.QtCore.QMetaObject object>

2.2.7 Icon module

Module containing classes for using and retriving icons files

class megascan_link.icon.**MegascanIcon**
Simple class for storing and retriving the Megascan Logo as a path to file or as a SDTexture instance

path = './megascan_logo.png'
Path to the Megascan Logo

Type str

2.2.8 Utilities module

Module containing utilities function for general usage

megascan_link.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

megascan_link.utilities.**getApp**() → sd.api.sdapplication.SDApplication
Helper function to retrive the SDApplication

Returns the substance designer application instance

Return type SDApplication

megascan_link.utilities.**getUiManager**()
Helper function to retrive the QtPythonUIManager

Returns return the current QtForPythonUIMgr instance

Return type UIMngr

2.2.9 Log module

This Module contains the logger facilities class

class megascan_link.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 setUpLogger()
Method used to setup the current logger instance

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

2.2.10 Module contents

class megascan_link.**Data**
Bases: object

"Dataclass" for storing plugin variables So the python garbage collector doesn't dispose them

settingDialog = None

socketThread = None

toolbar = None

toolbarAction = None

megascan_link.**createToolBarAction**()
function for create and setup the Megascan top toolbar icon for opening the plugin settings

megascan_link.**initializeSDPlugin**()
Main entry point of the plugin

megascan_link.**openSettings**()
function for setup and open the SettingsDialog

megascan_link.**uninitializeSDPlugin**()
Exit point of the plugin

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

O

P

R

S

T

U