# nodedge

***Release 0.2.0***

**May 03, 2020**

# Contents

Nodedge is a free and open-source software for graphical programming. Try it out and contribute to its development!

### Features

- Create your simulation easily thanks to a large variety of customizable built-in blocks.
- Access your models anywhere anytime. Nodedge is free for any purpose.
- Contribute to Nodedge development. The software is based on Python.

### Documentation

CHAPTER 1

# Introduction

Welcome to the Nodedge User's Manual. This manual contains information on using Nodedge software, including documentation for all development modules in the package and examples illustrating their use.

## 1.1 Overview

## 1.2 Some differences from Simulink

## 1.3 Installation

*nodedge* package can be installed using pip or the standard distutils/setuptools mechanisms.

To install using pip:

```
pip install nodedge
```

Alternatively, to use setuptools, first clone or download the source. To install in your default python interpreter, use:

```
git clone git@github.com:nodedge/nodedge.git
cd nodedge
python setup.py install --user
```

## 1.4 Getting started

There are two different ways to use the package. For the default interface, simply import the control package as follows:

```
>>> import nodedge
```

CHAPTER 2

# Library conventions

The nodedge library uses a set of standard conventions for the way that different types of standard information used by the library.

## 2.1 LTI system representation

Linear time invariant (LTI) systems are represented in nodedge in state space, transfer function, or frequency response data (FRD) form. Most functions in the toolbox will operate on any of these data types and functions for converting between compatible types is provided.

### 2.1.1 State space systems

$$\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

where u is the input, y is the output, and x is the state.

### 2.1.2 Transfer functions

$$G(s) = \frac{\text{num}(s)}{\text{den}(s)} = \frac{a_0 s^m + a_1 s^{m-1} + \cdots + a_m}{b_0 s^n + b_1 s^{n-1} + \cdots + b_n},$$

where n is generally greater than or equal to m (for a proper transfer function).

CHAPTER 3

# nodedge package

## 3.1 Subpackages

### 3.1.1 nodedge.blocks package

#### Submodules

#### nodedge.blocks.add_block

**class** nodedge.blocks.add_block.**AddBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 3**

**operationTitle = 'Add'**

**contentLabel = '+'**

**contentLabelObjectName = 'BlockBackground'**

**evalImplementation**()

#### nodedge.blocks.block

**class** nodedge.blocks.block.**Block**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.node.Node*

**iconPath = ''**

**operationTitle = 'Undefined'**

**operationCode = 0**

**contentLabel = ''**

**contentLabelObjectName = 'blockBackground'**

**GraphicsNodeClass**
alias of *nodedge.blocks.graphics_block.GraphicsBlock*

**GraphicsNodeContentClass**
alias of *nodedge.blocks.graphics_block_content.GraphicsBlockContent*

**initSettings**()
Initialize properties and sockets information.

**onInputChanged**(*socket: Optional[nodedge.socket.Socket] = None*)
Handle event associated with a change in this node's input edge. When it happens, this node and all its descendants are labelled as dirty.

**Parameters** **socket** (*Socket*) – reference to the changed *Socket*

**checkInputsValidity**()

**evalImplementation**()

**eval**(*index=0*)
Evaluate this node. This must be overridden. See *Evaluation* for more details.

**serialize**()
Serialization method to serialize this class data into `OrderedDict` which can be stored in memory or file easily.

**Returns** data serialized in `OrderedDict`

**Return type** `OrderedDict`

**deserialize**(*data*, *hashmap=None*, *restoreId=True*)
Deserialization method which take data in python `dict` format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (`dict`) – dictionary containing serialized data
- **hashmap** (`dict`) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (`bool`) – `True` if we are creating new sockets. `False` is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** `True` if deserialization was successful, `False` otherwise

**Return type** `bool`

**exception** nodedge.blocks.block.**EvaluationError**
Bases: `Exception`

**exception** nodedge.blocks.block.**MissInputError**
Bases: *nodedge.blocks.block.EvaluationError*

**exception** nodedge.blocks.block.**RedundantInputError**
Bases: *nodedge.blocks.block.EvaluationError*

### nodedge.blocks.block_config

**exception** nodedge.blocks.block_config.**BlockConfigException**
Bases: Exception

**exception** nodedge.blocks.block_config.**InvalidNodeRegistration**
Bases: *nodedge.blocks.block_config.BlockConfigException*

**exception** nodedge.blocks.block_config.**OperationCodeNotRegistered**
Bases: *nodedge.blocks.block_config.BlockConfigException*

nodedge.blocks.block_config.**associateOperationCodeWithBlock**(*operationCode*, *referenceClass*)

nodedge.blocks.block_config.**registerNode**(*operationCode*)

nodedge.blocks.block_config.**getClassFromOperationCode**(*operationCode*)

### nodedge.blocks.divide_block

**class** nodedge.blocks.divide_block.**DivideBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)
Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 6**

**operationTitle = 'Divide'**

**contentLabel = '/'**

**contentLabelObjectName = 'BlockBackground'**

**evalImplementation**()

### nodedge.blocks.graphics_block

**class** nodedge.blocks.graphics_block.**GraphicsBlock**(*node: Node*, *parent: Optional[PyQt5.QtWidgets.QGraphicsItem] = None*)
Bases: *nodedge.graphics_node.GraphicsNode*

**Parameters**

- **node** (*Node*) – reference to *Node*
- **parent** (Optional[QGraphicsItem]) – parent widget

**initSizes**()
Set up internal attributes like *width*, *height*, etc.

**initStyle**()
Initialize QObjects like QColor, QPen and QBrush.

**paint**(*painter*, *QStyleOptionGraphicsItem*, *widget=None*)
Paint the rounded rectangular *Node*.

### nodedge.blocks.graphics_block_content

**class** nodedge.blocks.graphics_block_content.**GraphicsBlockContent**(*node: Node*, *parent: Optional[PyQt5.QtWidgets.QWidget] = None*)

Bases: *nodedge.graphics_node_content.GraphicsNodeContent*

**Parameters**

- **node** (*Node*) – reference to the *Node*
- **parent** (*QWidget*) – parent widget

**Instance Attributes**

- **node** - reference to the *Node*
- **layout** - QLayout container

**initUI**()

Sets up layouts and widgets to be rendered in QDMGraphicsNode class.

### nodedge.blocks.graphics_input_block_content

**class** nodedge.blocks.graphics_input_block_content.**GraphicsInputBlockContent**(*node: Node*, *parent: Optional[PyQt5.QtWidg = None*)

Bases: *nodedge.graphics_node_content.GraphicsNodeContent*

**Parameters**

- **node** (*Node*) – reference to the *Node*
- **parent** (*QWidget*) – parent widget

**Instance Attributes**

- **node** - reference to the *Node*
- **layout** - QLayout container

**initUI**()

Sets up layouts and widgets to be rendered in QDMGraphicsNode class.

**serialize**()

Default serialization method.

It needs to be overridden for each node implementation.

**Return OrderedDict** Serialized data as ordered dictionary

**deserialize**(*data*, *hashmap=None*, *restoreId=False*)

Default deserialize method.

It needs to be overridden for each node implementation.

**Parameters**

- **data** (*dict*) – serialized data dictionary
- **hashmap** (*dict*) –
- **restoreId** (*bool*) – whether or not the id of the objects are restored

**Return bool** success status

**onEditingFinished**()

### nodedge.blocks.graphics_output_block_content

**class** nodedge.blocks.graphics_output_block_content.**GraphicsOutputBlockContent**(*node: Node*, *parent: Optional[PyQt5.QtW* = *None*)

Bases: *nodedge.graphics_node_content.GraphicsNodeContent*

**Parameters**

- **node** (*Node*) – reference to the *Node*
- **parent** (*QWidget*) – parent widget

**Instance Attributes**

- **node** - reference to the *Node*
- **layout** - QLayout container

**initUI**()

Sets up layouts and widgets to be rendered in QDMGraphicsNode class.

### nodedge.blocks.input_block

**class** nodedge.blocks.input_block.**InputBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 1**

**operationTitle = 'Input'**

**contentLabel = 'In'**

**contentLabelObjectName = 'InputBlockContent'**

**initInnerClasses**()

Set up graphics node and content widget.

**evalImplementation**()

### nodedge.blocks.multiply_block

**class** nodedge.blocks.multiply_block.**MultiplyBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 5**

**operationTitle = 'Multiply'**

**contentLabel = '*'**

**contentLabelObjectName = 'BlockBackground'**

**evalImplementation**()

### nodedge.blocks.output_block

**class** nodedge.blocks.output_block.**OutputBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 2**

**operationTitle = 'Output'**

**contentLabel = 'Out'**

**contentLabelObjectName = 'OutputBlockContent'**

**initInnerClasses**()

Set up graphics node and content widget.

**evalImplementation**()

### nodedge.blocks.subtract_block

**class** nodedge.blocks.subtract_block.**SubtractBlock**(*scene*, *inputSocketTypes=(2, 2)*, *outputSocketTypes=(1, )*)

Bases: *nodedge.blocks.block.Block*

**icon = '/home/docs/checkouts/readthedocs.org/user_builds/nodedge/checkouts/latest/node**

**operationCode = 4**

**operationTitle = 'Subtract'**

**contentLabel = '-'**

**contentLabelObjectName = 'BlockBackground'**

**evalImplementation**()

### Module contents

## 3.2 Submodules

### 3.2.1 nodedge.edge

Edge module containing *Edge* and *EdgeType* class.

**class** nodedge.edge.**EdgeType**
Bases: enum.IntEnum

Edge Type Constants

**STRAIGHT = 1**

**BEZIER = 2**

**class** nodedge.edge.**Edge**(*scene: Scene*, *startSocket: Optional[nodedge.socket.Socket] = None*, *endSocket: Optional[nodedge.socket.Socket] = None*, *edgeType: nodedge.edge.EdgeType = <EdgeType.BEZIER: 2>*)
Bases: *nodedge.serializable.Serializable*

Edge class.

The edge is the component connecting two *Node* s.

[NODE 1]——EDGE——[NODE 2]

**Parameters**

- **scene** (*Scene*) – Reference to the scene
- **startSocket** (*Socket*) – Reference to the starting socket
- **endSocket** (*Socket* | None) – Reference to the End socket or None
- **edgeType** (*EdgeType*) – Constant determining type of edge.

**Instance Attributes**

- **scene** - reference to the *Scene*
- **graphicsEdge**
- **Instance of *GraphicsEdge* subclass** handling graphical representation in the QGraphicsScene.

**sourceSocket**
Source socket.

**Getter** Return source *Socket*.

**Setter** Set source *Socket* safely.

**Type** *Socket*

**targetSocket**
Target socket

**Getter** Return target *Socket* or None if not set.

**Setter** Set target *Socket* safely.

**Type** *Socket* or None

**edgeType**
Edge type

**Getter** Get edge type constant for current *Edge*.

**Setter** Set new edge type. On background, create new *GraphicsEdge* child class if necessary, add this QGraphicsPathItem to the QGraphicsScene and update edge sockets positions.

**Type** *EdgeType*

**isSelected**
Property defining whether the edge is selected or not.

**Getter** Get selection state of the edge.

**Setter** Provide the safe selecting/deselecting operation. In the background it takes care about the flags, notifications and storing history for undo/redo.

**Type** bool

**updatePos**() → None
Update the internal *GraphicsEdge* positions according to the start and end *Socket*

**getOtherSocket**(*knownSocket: nodedge.socket.Socket*)
Return the opposite *Socket* on this *Edge*.

**Parameters** **knownSocket** (*Socket*) – Provide known *Socket* to be able to determine the opposite one

**Returns** The opposite socket on this *Edge*, eventually None.

**Return type** *Socket* or None

**removeFromSockets**()
Set start and end *Socket* to None

**remove**(*silentForSocket: Optional[nodedge.socket.Socket] = None*, *silent: bool = False*)
Safely remove this *Edge*.

Remove *GraphicsEdge* from the QGraphicsScene and it's reference to all other graphical elements. Notify previously connected *Node* (s) about this event.

Triggered Node Slots: - *onEdgeConnectionChanged()* - *onInputChanged()*

**Parameters**

- **silentForSocket** (Optional[*Socket*]) – Socket for whom the removal is silent
- **silent** (bool) – True if no events should be triggered during removing

**serialize**()
Serialization method to serialize this class data into OrderedDict which can be stored in memory or file easily.

**Returns** data serialized in OrderedDict

**Return type** OrderedDict

**deserialize**(*data*, *hashmap=None*, *restoreId=True*)
Deserialization method which take data in python dict format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (dict) – dictionary containing serialized data

- **hashmap** (dict) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (bool) – True if we are creating new sockets. False is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** True if deserialization was successful, False otherwise

**Return type** bool

### 3.2.2 nodedge.editor_widget

Editor widget module containing *EditorWidget* class.

**class** nodedge.editor_widget.**EditorWidget**(*parent=None*)

Bases: PyQt5.QtWidgets.QWidget

Default constructor.

**Parameters** **parent** (QWidget) – parent widget

**Instance Attributes**

- **filename** - currently graph's filename or None

**SceneClass**

alias of *nodedge.scene.Scene*

**initUI**()

Set up this *EditorWidget* with its layout, *Scene* and *GraphicsView*.

**hasName**

**Getter** Return if a file has been loaded in this *EditorWidget* or not.

**Return type** bool

**shortName**

**Getter** Return the short name of this *EditorWidget*.

**Return type** str

**userFriendlyFilename**

**Getter** Return the user friendly filename.

**Note:** This name is displayed as window title.

**Return type** str

**isModified**

**Getter** Has current *Scene* been modified?

**Return type** bool

**canUndo**

**Getter** Return whether previously executed operations are saved in history or not.

**Return type** bool

**canRedo**

**Getter** Return whether the history contains cancelled operations or not.

**Return type** bool

**selectedItems**

**Getter** Return *Scene*'s currently selected items.

**Return type** list[QGraphicsItem]

**hasSelectedItems**

**Getter** Return True if there is selected items in the nodedge.node_scene.Scene.

**Return type** bool

**updateTitle**() → None
Update the QMainWindow's title with the user friendly filename.

**newFile**() → None
Create a new file. Clear the scene and history, and reset filename.

**loadFile**(*filename: str*) → bool
Load serialized graph from JSON file.

**Parameters** **filename** (str) – file to load

**Returns** Operation success

**Return type** bool

**saveFile**(*filename: Optional[str] = None*) → bool
Save serialized graph to JSON file. When called with empty parameter, the filename is unchanged.

**Parameters** **filename** (str | None) – file to store the graph

**Returns** Operation success

**Return type** bool

**evalNodes**() → None
Evaluate all the nodes present in the scene.

**mouseReleaseEvent**(*ev: PyQt5.QtGui.QMouseEvent*) → None
Handle Qt's mouse's button release event.

**Parameters** **ev** (QMouseEvent) – Mouse release event

**mousePressEvent**(*ev: PyQt5.QtGui.QMouseEvent*) → None
Handle Qt's mouse's button press event.

**Parameters** **ev** (QMouseEvent) – Mouse press event

**addDebugContent**() → None
Testing method to put random QGraphicsItems and elements into QGraphicsScene

**addNodes**() → None
Testing method to create 3 *Node* connected by 2 *Edge*.

**addCustomNode**()
Testing method to create a custom Node with custom content

### 3.2.3 nodedge.editor_window

Editor window module containing *EditorWindow* class.

**class** nodedge.editor_window.**EditorWindow**(*parent: Optional[PyQt5.QtWidgets.QWidget] = None*)

Bases: PyQt5.QtWidgets.QMainWindow

*EditorWindow* class

The editor window is the base of the multi document interface (MDI) *MdiWindow*.

The application can be opened with the *EditorWindow* as main window, even if it is not the main use case.

**Instance Attributes**

- **name_company** - name of the company, used for permanent profile settings
- **name_product** - name of this App, used for permanent profile settings

**EditorWidgetClass**
alias of *nodedge.editor_widget.EditorWidget*

**currentEditorWidget**

**Getter** Get current *EditorWidget*

**Note:** The *EditorWindow* has only one *EditorWidget*. This method is overridden by the *MdiWindow* which may have several *EditorWidget*.

**Return type** *editor_widget*

**initUI**() → None
Set up this QMainWindow.

Create *EditorWidget*, Actions and Menus

**createStatusBar**() → None
Create Status bar and connect to *GraphicsView*'s scenePosChanged event.

**createActions**() → None
Create basic *File* and *Edit* actions.

**createMenus**() → None
Create Menus for *File* and *Edit*.

**createFileMenu**()
Create *File* Menu.

**createEditMenu**()
Create *Edit* Menu.

**sizeHint**(*self*) → QSize

**updateTitle**() → None
Update window title according to the name of the file currently opened.

**onClipboardChanged**() → None

**OnScenePosChanged**(*x: float*, *y: float*)
Handle event when cursor position changed on the *Scene*. :param x: new cursor x position :type x: float :param y: new cursor y position :type y: float

**newFile**()
Open a clean new file in the window's editor.

Confirmation is asked to the user if there are unsaved changes.

**openFile**(*filename*)
Open a file in the window's editor from its filename.

Confirmation is asked to the user if there are unsaved changes.

**Parameters** **filename** (str) – absolute path and filename of the file to open.

**saveFile**()
Save serialized JSON version of the currently opened file, in a JSON file based on the editor's filename.

**saveFileAs**()
Save serialized JSON version of the currently opened file, allowing the user to choose the filename via a QFileDialog.

**closeEvent**(*event: PyQt5.QtGui.QCloseEvent*) → None
Close the window.

Confirmation is asked to the user if there are unsaved changes.

**quit**()

**undo**() → None
Undo last operation.

**redo**() → None
Redo previously cancelled operation.

**delete**() → None
Delete selected items.

**cut**() → None
Cut to clipboard selected items.

**copy**() → None
Copy to clipboard selected items.

**paste**()
Paste from clipboard, creating items after deserialization.

**static getFileDialogDirectory**() → str
Returns starting directory for QFileDialog file open/save

**Returns** starting directory for QFileDialog file open/save

**Return type** str

**static getFileDialogFilter**()
Returns str standard file open/save filter for QFileDialog

**Returns** standard file open/save filter for QFileDialog

**Return type** str

**maybeSave**()
If current *Scene* is modified, ask a dialog to save the changes.

**Returns** True if the action calling this method is allowed to continue. False if we should cancel operation.

**Return type** bool

**readSettings**()
Read the permanent profile settings for this application.

**writeSettings**()
Write the permanent profile settings for this application.

### 3.2.4 nodedge.graphics_cut_line

Graphics cut line module containing *GraphicsCutLine* class.

**class** nodedge.graphics_cut_line.**GraphicsCutLine**(*parent=None*)
Bases: PyQt5.QtWidgets.QGraphicsItem

*GraphicsCutLine* class

Cutting Line used for cutting multiple *Edges* with one stroke

**Parameters** **parent** (QWidget) – parent widget

**boundingRect**() → PyQt5.QtCore.QRectF
Define Qt' bounding rectangle

**shape**() → PyQt5.QtGui.QPainterPath
Calculate the QPainterPath object from list of line points.

**Returns** shape function returning QPainterPath representation of Cutting Line

**Return type** QPainterPath

**paint**(*painter: PyQt5.QtGui.QPainter*, *option: PyQt5.QtWidgets.QStyleOptionGraphicsItem*, *widget: Optional[PyQt5.QtWidgets.QWidget] = None*) → None
Paint the cut line

### 3.2.5 nodedge.graphics_edge

Graphics edge module containing *GraphicsEdge*, *GraphicsEdgeDirect* and *GraphicsEdgeBezier* classes.

**class** nodedge.graphics_edge.**GraphicsEdge**(*edge: Edge*, *parent: Optional[PyQt5.QtWidgets.QGraphicsItem] = None*)
Bases: PyQt5.QtWidgets.QGraphicsPathItem

*GraphicsEdge* class

The graphics edge is the graphical representation of the *Edge*.

**Parameters**

- **edge** (*Edge*) – reference to *Edge*
- **parent** (Optional[QGraphicsItem]) – parent widget

**selectedState**

**Getter** Return whether the edge is selected or not.

**Setter** Set the selection state of the edge.

**Type** bool

**sourcePos**

**Getter** Return the edge's source position.

**Setter** Set the edge's source position.

**Type** QPointF

**targetPos**

**Getter** Return the edge's target position.

**Setter** Set the edge's target position.

**Type** QPointF

**initUI**()
Set up this QGraphicsPathItem

**initStyle**()
Initialize QObject like QColor, QPen and QBrush

**boundingRect**()
Define Qt' bounding rectangle

**onSelected**()
Slot called when the edge has just been selected.

**mouseReleaseEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneMouseEvent*) → None
Overridden Qt's slot to handle mouse release on the edge.

**Parameters** **event** (QGraphicsSceneMouseEvent) – Qt's mouse release event

**hoverEnterEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneHoverEvent*) → None
Overridden Qt's slot to handle mouse hovering on the edge.

**Parameters** **event** (QGraphicsSceneHoverEvent) – Qt's mouse hover event

**hoverLeaveEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneHoverEvent*) → None
Overridden Qt's slot to handle mouse hovering's end on the edge.

**Parameters** **event** (QGraphicsSceneHoverEvent) – Qt's mouse hover event

**shape**() → PyQt5.QtGui.QPainterPath
Returns QPainterPath representation of the edge.

**Returns** graphical path

**Return type** QPainterPath

**paint**(*painter*, *QStyleOptionGraphicsItem*, *widget=None*)
Qt's overridden method to paint the edge.

**Note:** The path is calculated in *calcPath()* method.

**calcPath**() → PyQt5.QtGui.QPainterPath
Compute the graphical path between *sourcePos* and ~*nodedge.graphics_edge.GraphicsEdge.targetPos*.

**Warning:** This method needs to be overridden.

**Returns** The computed path

**Return type** QPainterPath

**intersectsWith**(*p1: PyQt5.QtCore.QPointF*, *p2: PyQt5.QtCore.QPointF*) → bool
Compute if the edge's path intersects with line between points given as argument.

**Parameters**

- **p1** (QPointF) – first point
- **p2** (QPointF) – second point

**Returns** True if this edge's path intersects with the line between p1 and p2

**Return type** bool

**class** nodedge.graphics_edge.**GraphicsEdgeDirect**(*edge: Edge*, *parent: Optional[PyQt5.QtWidgets.QGraphicsItem] = None*)

Bases: *nodedge.graphics_edge.GraphicsEdge*

Graphics Edge Direct class, with straight line path between *sourcePos* and *targetPos*

**Parameters**

- **edge** (*Edge*) – reference to *Edge*
- **parent** (Optional[QGraphicsItem]) – parent widget

**calcPath**() → PyQt5.QtGui.QPainterPath
Compute a straight line path between *sourcePos* and *~nodedge.graphics_edge.GraphicsEdge.targetPos*.

**Returns** The computed path

**Return type** QPainterPath

**class** nodedge.graphics_edge.**GraphicsEdgeBezier**(*edge: Edge*, *parent: Optional[PyQt5.QtWidgets.QGraphicsItem] = None*)

Bases: *nodedge.graphics_edge.GraphicsEdge*

Graphics Edge Bezier class, with Bezier line path between *sourcePos* and *targetPos*

**Parameters**

- **edge** (*Edge*) – reference to *Edge*
- **parent** (Optional[QGraphicsItem]) – parent widget

**calcPath**() → PyQt5.QtGui.QPainterPath
Compute a Bezier curve path between *sourcePos* and *targetPos*.

**Returns** The computed path

**Return type** QPainterPath

### 3.2.6 nodedge.graphics_node

Graphics node module containing *GraphicsNode* class.

**class** nodedge.graphics_node.**GraphicsNode**(*node: Node*, *parent: Optional[PyQt5.QtWidgets.QGraphicsItem] = None*)

Bases: PyQt5.QtWidgets.QGraphicsItem

*Node* class

The graphics node is the graphical representation of a node.

**Parameters**

- **node** (*Node*) – reference to *Node*
- **parent** (Optional[QGraphicsItem]) – parent widget

**title**
Title of this *GraphicsNode*.

**Getter** Return current *GraphicsNode* title

**Setter** Store and make visible the new title

**Type** str

**selectedState**

**content**

**Getter** Return reference to *GraphicsNodeContent*

**Return type** *GraphicsNodeContent*

**initUI**() → None
Set up this QGraphicsItem.

**initStyle**() → None
Initialize QObjects like QColor, QPen and QBrush.

**initSizes**() → None
Set up internal attributes like *width*, *height*, etc.

**initTitle**() → None
Set up the title Graphics representation: font, color, position, etc.

**initContent**() → None
Set up the *GraphicsNodeContentProxy* to have a container for *GraphicsNodeContent*.

**boundingRect**()
Define Qt' bounding rectangle.

**paint**(*painter*, *QStyleOptionGraphicsItem*, *widget=None*)
Paint the rounded rectangular *Node*.

**mouseMoveEvent**(*event*)
Override Qt's event to detect that we moved this .

**mouseReleaseEvent**(*event*)
Handle Qt's event when we move, select or deselect this *GraphicsNode*.

**hoverEnterEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneHoverEvent*) → None
Handle Qt's hover event. It adds a highlighting boundary around this *GraphicsNode*.

**hoverLeaveEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneHoverEvent*) → None
Handle Qt's hover effect.

**mouseDoubleClickEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneMouseEvent*)
Qt's overridden event for doubleclick. Resend to *onDoubleClicked()*

**onSelected**()
Handle when the node has been selected.

### 3.2.7 nodedge.graphics_node_content

Graphics node content module containing the *GraphicsNodeContent* class.

**class** nodedge.graphics_node_content.**GraphicsNodeContent**(*node: Node*, *parent: Optional[PyQt5.QtWidgets.QWidget] = None*)

Bases: PyQt5.QtWidgets.QWidget, *nodedge.serializable.Serializable*

*GraphicsNodeContent* class.

Base class for representation of the Node's graphics content. This class also provides layout for other widgets inside of a *Node*.

**Parameters**

- **node** (*Node*) – reference to the *Node*
- **parent** (*QWidget*) – parent widget

**Instance Attributes**

- **node** - reference to the *Node*
- **layout** - QLayout container

**initUI**()

Sets up layouts and widgets to be rendered in QDMGraphicsNode class.

**setEditingFlag**(*value: bool*) → None

---

**Note:** If you are handling keyPress events by default Qt Window's shortcuts and QActions, you will not probably need to use this method

---

Helper function which sets editingFlag inside *GraphicsView* class.

This is a helper function to handle keys inside nodes with QLineEdits or QTextEdits (you can use overridden *TextEdit* class) and with QGraphicsView class method keyPressEvent.

**Parameters** **value** (bool) – new value for editing flag

**serialize**() → collections.OrderedDict

Default serialization method.

It needs to be overridden for each node implementation.

**Return OrderedDict** Serialized data as ordered dictionary

**deserialize**(*data: dict*, *hashmap: Optional[dict] = None*, *restoreId: bool = False*) → bool

Default deserialize method.

It needs to be overridden for each node implementation.

**Parameters**

- **data** (*dict*) – serialized data dictionary
- **hashmap** (*dict*) –
- **restoreId** (*bool*) – whether or not the id of the objects are restored

**Return bool** success status

**class** nodedge.graphics_node_content.**TextEdit**
Bases: PyQt5.QtWidgets.QTextEdit

**Note:** This class is example of QTextEdit modification to be able to handle *Delete* key with overridden Qt's keyPressEvent (when not using QActions in menu or toolbar)

overridden QTextEdit which sends notification about being edited to parent's container *GraphicsNodeContent*

**focusInEvent**(*event: PyQt5.QtGui.QFocusEvent*) → None
Example of overridden focusInEvent to mark start of editing.

**Parameters** **event** (*QFocusEvent*) – Qt's focus event

**focusOutEvent**(*event: PyQt5.QtGui.QFocusEvent*) → None
Example of overridden focusOutEvent to mark end of editing

**Parameters** **event** (*QFocusEvent*) – Qt's focus event

**class** nodedge.graphics_node_content.**GraphicsNodeContentProxy**(*graphicsNodeParent: GraphicsNode*)
Bases: PyQt5.QtWidgets.QGraphicsProxyWidget

### 3.2.8 nodedge.graphics_node_title_item

Graphics node title item module containing *GraphicsNodeTitleItem* class.

**class** nodedge.graphics_node_title_item.**GraphicsNodeTitleItem**(*graphicsNodeParent: GraphicsNode*)
Bases: PyQt5.QtWidgets.QGraphicsTextItem

### 3.2.9 nodedge.graphics_scene

Graphics scene module containing *GraphicsScene* class.

**class** nodedge.graphics_scene.**GraphicsScene**(*scene: Scene*, *parent: Optional[PyQt5.QtWidgets.QWidget] = None*)
Bases: PyQt5.QtWidgets.QGraphicsScene

*Scene* class

The graphics scene contains the background grid.

**Parameters**

- **scene** (*Scene*) – reference to the *Scene*
- **parent** (*QWidget*) – parent widget

**itemSelected**
pyqtSignal emitted when some item is selected in the *Scene*

**itemsDeselected**
pyqtSignal emitted when items are deselected in the *Scene*

**initUI**()
Set up this QGraphicsScene

**initStyle**()
Initialize QObjects like QColor, QPen and QBrush

**initSizes**()
Set up internal attributes like *grid_size*, *scene_width* and *scene_height*.

**setScene**(*width*, *height*)
Set *width* and *height* of the graphics scene.

**drawBackground**(*painter*, *rectangle*)
Draw background scene grid.

**dragMoveEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneDragDropEvent*) → None
Handle Qt's mouse's drag move event.

**Parameters** **event** (QGraphicsSceneDragDropEvent) – Mouse release event

**mousePressEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneMouseEvent*) → None
Handle Qt's mouse's button press event.

**Parameters** **event** (QGraphicsSceneMouseEvent.py) – Mouse release event

**mouseReleaseEvent**(*event: PyQt5.QtWidgets.QGraphicsSceneMouseEvent*)
Handle Qt's mouse's button release event.

**Parameters** **event** (QGraphicsSceneMouseEvent.py) – Mouse release event

### 3.2.10 nodedge.graphics_socket

Graphics socket module containing *GraphicsSocket* class.

**class** nodedge.graphics_socket.**GraphicsSocket**(*socket: Socket*)
Bases: PyQt5.QtWidgets.QGraphicsItem

*GraphicsSocket* class.

The graphics socket is the graphical representation of the *Socket*.

**Parameters** **socket** (*Socket*) – reference to *Socket*

**initUI**() → None
Setup this QGraphicsItem.

**initStyle**() → None
Initialize QObjects like QColor, QPen and QBrush.

**initSizes**() → None
Set up internal attributes like *width*, *height*, etc.

**socketType**

**static getSocketColor**(*key*)
Returns the QColor for this key.

**updateSocketType**()
Change the Socket Type.

**paint**(*painter*, *QStyleOptionGraphicsItem*, *widget=None*)
Paint a circle.

**boundingRect**()
Define Qt's bounding rectangle.

### 3.2.11 nodedge.graphics_view

Graphics View module containing *GraphicsView* and *DragMode* classes.

**class** nodedge.graphics_view.**DragMode**
Bases: enum.IntEnum

*DragMode* class.

**NOOP = 1**
Mode representing ready state

**EDGE_DRAG = 2**
Mode representing when we drag edge state

**EDGE_CUT = 3**
Mode representing when we draw a cutting edge

nodedge.graphics_view.**EDGE_START_DRAG_THRESHOLD = 10**
Distance when click on socket to enable *Drag Edge*

**class** nodedge.graphics_view.**GraphicsView**(*graphicsScene: nodedge.graphics_scene.GraphicsScene*, *parent: Optional[PyQt5.QtWidgets.QWidget] = None*)
Bases: PyQt5.QtWidgets.QGraphicsView

*GraphicsView* class.

**Parameters**

- **graphicsScene** (*GraphicsScene*) – reference to the *GraphicsScene*
- **parent** (Optional[QWidget]) – parent widget

**scenePosChanged**
pyqtSignal emitted when cursor position on the *Scene* has changed

**initUI**()
Set up this *GraphicsView*.

**dragEnterEvent**(*event: PyQt5.QtGui.QDragEnterEvent*) → None
Handle Qt's mouse's drag enter event.

Call all the listeners of that event.

**Parameters** **event** (QDragEnterEvent) – Mouse drag enter event

**dropEvent**(*event: PyQt5.QtGui.QDropEvent*) → None
Handle Qt's mouse's drop event.

Call all the listeners of that event.

**Parameters** **event** (QDropEvent) – Mouse drop event

**addDragEnterListener**(*callback: Callable[PyQt5.QtGui.QDragEnterEvent, None]*)
Register callback for *Drag Enter* event.

**Parameters** **callback** – callback function

**addDropListener**(*callback: Callable[PyQt5.QtGui.QDropEvent, None]*)
Register callback for *Drop* event.

**Parameters** **callback** – callback function

**mousePressEvent**(*event: PyQt5.QtGui.QMouseEvent*)
Dispatch Qt's *mousePressEvent* to corresponding function below.

**mouseReleaseEvent**(*event: PyQt5.QtGui.QMouseEvent*)
Dispatch Qt's mouseReleaseEvent to corresponding function below.

**leftMouseButtonPress**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when the left mouse button is pressed.

**leftMouseButtonRelease**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when left mouse button is released.

**middleMouseButtonPress**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when middle mouse button is pressed.

**middleMouseButtonRelease**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when middle mouse button is released.

**rightMouseButtonPress**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when right mouse button was pressed

**rightMouseButtonRelease**(*event: PyQt5.QtGui.QMouseEvent*)
Handle when right mouse button is released.

**dragEdgeStart**(*item*)
Handle the start of dragging an *Edge* operation.

**dragEdgeEnd**(*item: PyQt5.QtWidgets.QGraphicsItem*)
Handle the end of dragging an *Edge* operation.

**Parameters** **item** (QGraphicsItem) – Item in the *Graphics Scene* where we ended dragging an *Edge*

**Returns** True is the operation is a success, false otherwise.

**Return type** bool

**mouseMoveEvent**(*event: PyQt5.QtGui.QMouseEvent*) → None
Overridden Qt's mouseMoveEvent handling Scene/View logic

**Parameters** **event** (QMouseEvent.py) – Qt's mouse event

**keyPressEvent**(*event: PyQt5.QtGui.QKeyEvent*) → None
Handle key shortcuts, for example to display the scene's history in the console.

**Parameters** **event** (QKeyEvent.py) – Qt's Key event

**wheelEvent**(*event*)
Overridden Qt's wheelEvent. This handles zooming.

**cutIntersectingEdges**()
Compare which GraphicsCutLine and delete them safely.

**deleteSelected**()
Shortcut for safe deleting every object selected in the *Scene*.

**getItemAtClick**(*event: PyQt5.QtGui.QMouseEvent*)
Return the object on which the user clicked/released the mouse button.

**Parameters** **event** (QMouseEvent.py) – Qt's mouse or key event

**Returns** Graphical item present at the clicked/released position.

**Return type** QGraphicsItem | None

**distanceBetweenClickAndReleaseIsOff**(*event*)
Measure if we are too far from the last mouse button click scene position. This is used for detection if the release is too far after the user clicked on a *Socket*

**Parameters** **event** (QMouseEvent.py) – Qt's mouse event

**Returns** True if we released too far from where we clicked before, False otherwise.

**Return type** bool

**static debugModifiers**(*event*)
Get the name of the pressed modifier.

**Returns** "CTRL" / "SHIFT" / "ALT"

**Return type** str

### 3.2.12 nodedge.mdi_widget

Editor widget module containing *EditorWidget* class.

**class** nodedge.mdi_widget.**MdiWidget**
Bases: *nodedge.editor_widget.EditorWidget*

*MdiWidget* class.

The mdi widget represents a sub-window of the *MdiWindow*.

**initNewNodeActions**()
Add all available blocks in the *NodeListWidget*.

**initNodesContextMenu**()
Create a context menu containing all the nodes available, so that the user can quickly create a new block by right clicking on the *Scene*.

**static getNodeClassFromData**(*data*)
Get the node class associated with operation present in data.

**Parameters** **data** – serialized *Node* containing the operation code

**Type** dict

**Returns** class of the node associated with the operation code in data, node class in case of failure.

**Return type** Node class

**addCloseEventListener**(*callback: Callable[None]*)
Register callback for *Has Been Modified* event

**Parameters** **callback** (*Callable[[], None]*) – callback function

**closeEvent**(*event: PyQt5.QtGui.QCloseEvent*) → None
Handle Qt's close event.

Make sure changes have been saved before closing the widget.

**Parameters** **event** (QCloseEvent.py) – Qt's close event, the user may have clicked on the close button, or pressed CTRL+W

**onNodeDragEnter**(*event: PyQt5.QtGui.QDragEnterEvent*)
Handle node drag enter event.

When a node is dragged from the *NodeListWidget*, its logo is displayed above the scene, near the location of the mouse.

**Parameters** **event** – the Qt's drag event event, containing the mime data of the node being dragged

**Returns** QDragEnterEvent

**onNodeDrop**(*event: PyQt5.QtGui.QDropEvent*)
Handle node drop event.

When the node is dropped, an instance of it is created near at the mouse location, displayed by its *GraphicsNode*.

**Parameters** **event** (*QDropEvent*) – the Qt's drop event, containing the mime data of the node being dropped.

**contextMenuEvent**(*event: PyQt5.QtGui.QContextMenuEvent*)
Handle Qt's context menu event.

**Parameters** **event** (QContextMenuEvent.py) – the Qt's context menu event, happening when the user right clicks on the *GraphicsScene*

**handleNodeContextMenu**(*event: PyQt5.QtGui.QContextMenuEvent*)
Handle Qt's context menu event when the user has clicked on a node.

**Parameters** **event** – Qt's context menu event, happening when the users

**Returns** QContextMenuEvent

**static determineTargetSocketOfNode**(*wasDraggedFlag: bool*, *newNode: nodedge.node.Node*) → Optional[nodedge.socket.Socket]

**finishNewNodeState**(*newNode*)

**handleNewNodeContextMenu**(*event*)
Handle context menu event when the users has right clicked on an empty space.

Show all available nodes available in a list context menu, so that the users can quickly create a new one.

**Parameters** **event** (QContextMenuEvent.py) – the Qt's context menu event, happening when the user right clicks on the *GraphicsScene*

**handleEdgeContextMenu**(*event*)
Handle Qt's context menu when the user has right clicked on an *GraphicsEdge*

**Parameters** **event** (QContextMenuEvent) – the Qt's context menu event, happening when the user right clicks on the *GraphicsScene*

**mouseReleaseEvent**(*event: PyQt5.QtGui.QMouseEvent*)
Handle Qt's mouse release event.

**Parameters** **event** – Qt's mouse release event

**Returns** QMouseEvent

### 3.2.13 nodedge.mdi_window

Multi Document Interface window module containing *MdiWindow* class.

**class** nodedge.mdi_window.**MdiWindow**
Bases: *nodedge.editor_window.EditorWindow*

*MdiWindow* class.

The mdi window is the main window of Nodedge.

**currentEditorWidget**
Property representing the *EditorWidget* of the active sub-window.

**Note:** This property cannot be set.

**Type** *EditorWidget*

**initUI**()
Set up this QMainWindow.

Create the mdi area, actions and menus

**createStatusBar**()
Create the status bar describing Nodedge status and the mouse position.

**createActions**()
Create *File*, *Edit* and *About* actions.

**createToolBars**()
Create the *File* and *Edit* toolbar containing few of their menu actions.

**createMenus**()
Create *Window* and *Help* menus.

*Window* menu allows to navigate between the sub-windows. *Help* menu allows to display know more about Nodedge.

**createHelpMenu**()

**createWindowMenu**()

**updateMenus**()
Update menus accordingly to the presence or not of sub-window in the editor, enabling and disabling file manipulation actions, for example.

**updateFileMenu**()
Update file menu.

**updateWindowMenu**()
Update window menu.

**updateEditMenu**()
Update edit menu.

**onSubWindowClosed**(*widget*, *event*)

**findMdiSubWindow**(*filename*)

**setActiveSubWindow**(*window*)

**createNodesDock**()

**createHistoryDock**()

**createSceneItemsDock**()

**closeEvent**(*event*)
Close the window.

Confirmation is asked to the user if there are unsaved changes.

**newFile**()
Open a clean new file in the window's editor.

Confirmation is asked to the user if there are unsaved changes.

**openFile**(*filenames*)
Open a file in the window's editor from its filename.

Confirmation is asked to the user if there are unsaved changes.

**Parameters** **filename** (str) – absolute path and filename of the file to open.

**about**()

**onNodesToolbarTriggered**()

**addCurrentEditorWidgetChangedListener**(*callback*)

**onSubWindowActivated**()

### 3.2.14 nodedge.node

Node module containing *Node* class.

**class** nodedge.node.**NodesAndSockets**(*nodes*, *sockets*)
Bases: object

**class** nodedge.node.**Node**(*scene: Scene*, *title: str = 'Undefined node'*, *inputSocketTypes: Collection[int] = ()*, *outputSocketTypes: Collection[int] = ()*)
Bases: *nodedge.serializable.Serializable*

*Node* class representing a node in the *Scene*.

**Parameters**

- **scene** (*Scene*) – reference to the *Scene*
- **title** (str) – node title shown in scene
- **inputSocketTypes** (Collection[int]) – list of types of the input *Sockets*
- **outputSocketTypes** (Collection[int]) – list of types of the output *Sockets*

**GraphicsNodeClass**
alias of *nodedge.graphics_node.GraphicsNode*

**GraphicsNodeContentClass**
alias of *nodedge.graphics_node_content.GraphicsNodeContent*

**SocketClass**
alias of *nodedge.socket.Socket*

**initInnerClasses**() → None
Set up graphics node and content widget.

**initSettings**() → None
Initialize properties and sockets information.

**initSockets**(*inputs: Collection[int], outputs: Collection[int], reset: bool = True*) → None
Create input and output sockets.

**Parameters**

- **inputs** (Collection[int]) – list of types of the input *Sockets*. Every type is associated with a int
- **outputs** (Collection[int]) – list of types of the input *Sockets*
- **reset** (bool) – if True destroy and remove old *Sockets*

**onEdgeConnectionChanged**(*newEdge: nodedge.edge.Edge*) → None
Handle event associated with a change in any of the connections (*Edge*). Currently unused.

**Parameters** **newEdge** (*Edge*) – reference to the changed *Edge*

**onInputChanged**(*socket: nodedge.socket.Socket*)
Handle event associated with a change in this node's input edge. When it happens, this node and all its descendants are labelled as dirty.

**Parameters** **socket** (*Socket*) – reference to the changed *Socket*

**title**
Title shown in the scene.

**Getter** return current node title

**Setter** set node title and pass it to the graphical node

**Return type** `str`

**pos**
Retrieve node's position in the scene

**Returns** node position

**Return type** `QPointF`

**isDirty**
Property stating whether or not this node is marked as *Dirty*, i.e. the node has not been evaluated since last node's input/output change.

**Getter** `True` if this node is marked as *Dirty*, `False` otherwise

**Setter** set the dirtiness status of this node

**Type** `bool`

**isInvalid**
Property stating whether or not this node is marked as *Invalid*, i.e. the node has been evaluated since last node's input/output change, but the evaluation was inconsistent.

**Getter** `True` if this node is marked as *Invalid*, `False` otherwise

**Setter** set the validity status of this node

**Type** `bool`

**isSelected**
Retrieve graphics node selection status.

**socketPos**(*index: int*, *location: int*, *countOnThisSide: int = 1*) → PyQt5.QtCore.QPointF
Get the relative *x, y* position of a *Socket*. This is used for placing the *GraphicsSocket* on *Graphics Node*.

**Parameters**

- **index** (`int`) – Order number of the Socket. (0, 1, 2, ...)
- **location** (*SocketLocation*) – *Socket location constant* describing where the Socket is located
- **countOnThisSide** (`int`) – Total number of Sockets on this *Socket Position*

**Returns** Position of described Socket on the *Node*

**Return type** `QPointF`

**updateConnectedEdges**()
Refresh positions of all connected *Edges*. It is used for updating graphical edges.

**remove**()
Safely remove this node.

**onMarkedDirty**()
Called when this *Node* has been marked as *Dirty*. This method must be overridden.

**markChildrenDirty**(*newValue: bool = True*) → None
Mark the children of this node to be *Dirty*. Children are first-level descendants. Note: it does not apply to this node.

**Parameters** **newValue** (bool) – True if children should be *Dirty*, False to un-dirty them.

**markDescendantsDirty**(*newValue: bool = True*) → None

**Mark all-level descendants of this *Node* to be *Dirty*.** Note: it does not apply to this node.

**Parameters** **newValue** (bool) – True if descendants should be *Dirty*, False to un-dirty them.

**onMarkedInvalid**() → None
Called when this node has been marked as *Invalid*. This method must be overridden.

**markChildrenInvalid**(*newValue: bool = True*) → None
Mark children of this node as *Invalid*. Children are first-level descendants. Note: it does not apply to this node.

**Parameters** **newValue** (bool) – True if children should be *Invalid*, False to make them valid.

**markDescendantsInvalid**(*newValue: bool = True*) → None
Mark descendants of this node as *Invalid*. Note: it does not apply to this node.

**Parameters** **newValue** (bool) – True if descendants should be *Invalid*, False to make descendants valid.

**eval**(*index=0*) → float
Evaluate this node. This must be overridden. See *Evaluation* for more details.

**evalChildren**() → None
Evaluate children of this node

**getChildrenNodes**() → List[nodedge.node.Node]
Retrieve all children connected to this node outputs.

**Returns** list of *Nodes* connected to this node from all outputs

**Return type** List[*Node*]

**inputNodesAt**(*index: int*) → List[nodedge.node.Node]
Get **all** nodes connected to the input specified by *index*.

**Parameters** **index** (int) – order number of the input socket

**Returns** all *Node* instances which are connected to the specified input or [] if there is no connection or index is out of range.

**Return type** List[*Node*]

**inputNodeAt**(*index: int*) → Optional[nodedge.node.Node]
Get the **first** node connected to the input specified by *index*.

**Parameters** **index** (int) – order number of the input socket

**Returns** *Node* which is connected to the specified input or None if there is no connection or index is out of range

**Return type** *Node*

**inputNodeAndSocketAt**(*index*)

**outputNodesAt**(*index: int*) → List[nodedge.node.Node]
Get **all** nodes connected to the output specified by *index*.

**Parameters** **index** (int) – order number of the output socket

**Returns** all *Node* instances which are connected to the specified output or [] if there is no connection or index is out of range.

**Return type** List[*Node*]

**serialize**() → collections.OrderedDict
Serialization method to serialize this class data into OrderedDict which can be stored in memory or file easily.

**Returns** data serialized in OrderedDict

**Return type** OrderedDict

**deserialize**(*data*, *hashmap=None*, *restoreId=True*) → bool
Deserialization method which take data in python dict format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (dict) – dictionary containing serialized data
- **hashmap** (dict) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (bool) – True if we are creating new sockets. False is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** True if deserialization was successful, False otherwise

**Return type** bool

**onDoubleClicked**(*event: PyQt5.QtWidgets.QGraphicsSceneMouseEvent*) → None
Callback when the *GraphicsNode* is double clicked.

**Parameters** **event** (QMouseEvent) – Qt double click event

**getNodeContentClass**()
Returns class representing node content.

**getGraphicsNodeClass**()
Returns class representing graphics node.

**getSocketScenePosition**(*socket: nodedge.socket.Socket*) → PyQt5.QtCore.QPointF
Get absolute *Socket* position in the *Socket*.

**Parameters** **socket** – The socket from which we want to get the position

**Returns** *Socket*'s scene position

**Return type** QPointF

**addSelectedListener**(*callback*)

### 3.2.15 nodedge.node_list_widget

Node list widget module containing *NodeListWidget* class.

**class** nodedge.node_list_widget.**NodeListWidget**(*parent: Optional[PyQt5.QtWidgets.QWidget] = None*)

Bases: PyQt5.QtWidgets.QListWidget

Node list widget class.

The list widget contains the declaration of all the available nodes.

**Parameters** **parent** (QWidget | None) – Qt's widget parent

**initUI**() → None
Set up this *NodeListWidget* with its icon and *Node*.

**addNodes**() → None
Add available *Node* s in the list widget.

**addNode**(*name*, *iconPath: Optional[str] = None*, *operationCode: int = 0*)
Add a *Node* in the list widget.

**startDrag**(**args*, ***kwargs*) → None
Serialize data when a user start dragging a node from the list, to be able to instantiate it later.

### 3.2.16 nodedge.scene

Scene module containing *Scene*.

**class** nodedge.scene.**Scene**

Bases: *nodedge.serializable.Serializable*

*Scene* class

**Instance Attributes**

- **nodes** - list of *Nodes* in this *Scene*
- **edges** - list of *Edges* in this *Scene*
- **history** - Instance of *SceneHistory*
- **clipboard - Instance of** *SceneClipboard*
- **scene_width** - width of this *Scene* in pixels
- **scene_height** - height of this *Scene* in pixels

**isModified**
Has this *Scene* been modified?

**Getter** True if the *Scene* has been modified

**Setter** set new state. Triggers *Has Been Modified* event

**Type** bool

**lastSelectedItems**

**selectedItems**
Returns currently selected Graphics Items

**Returns** list of QGraphicsItems

**Return type** list[QGraphicsItem]

**view**
Shortcut for returning *Scene* `QGraphicsView`

**Returns** `QGraphicsView` attached to the *Scene*

**Return type** `QGraphicsView`

**silentSelectionEvents**
” If this property is true, do not trigger onItemSelected when an item is selected

**Returns** True is onItemSelected is not triggered when an item is selected

**Return type** `bool`

**onItemSelected**(*silent: bool = False*)
Handle Item selection and trigger event *Item Selected*

**Parameters** **silent** (`bool`) – If `True` scene's onItemSelected won't be called and history stamp not stored.

**onItemsDeselected**(*silent: bool = False*)
Handle Items deselection and trigger event *Items Deselected*

**Parameters** **silent** (`bool`) – If `True` scene's onItemsDeselected won't be called and history stamp not stored.

**doDeselectItems**(*silent: bool = False*) → None
Deselects everything in scene

**Parameters** **silent** (`bool`) – If `True` scene's onItemsDeselected won't be called

**addHasBeenModifiedListener**(*callback: Callable[None]*)
Register callback for *Has Been Modified* event

**Parameters** **callback** (`Callable[[], None]`) – callback function

**addItemSelectedListener**(*callback: Callable[None]*)
Register callback for *Item Selected* event

**Parameters** **callback** (`Callable[[], None]`) – callback function

**addItemsDeselectedListener**(*callback: Callable[None]*)
Register callback for *Items Deselected* event

**Parameters** **callback** – callback function

**addDragEnterListener**(*callback: Callable[PyQt5.QtGui.QDragEnterEvent, None]*)
Register callback for *Drag Enter* event

**Parameters** **callback** – callback function

**addDropListener**(*callback: Callable[PyQt5.QtGui.QDropEvent, None]*)
Register callback for *Drop* event

**Parameters** **callback** – callback function

**resetLastSelectedStates**()
Resets internal *selected flags* in all *Nodes* and *Edges* in the *Scene*

**addNode**(*node: nodedge.node.Node*)
Add *Node* to this *Scene*

**Parameters** **node** (*Node*) – *Node* to be added to this *Scene*

**addEdge**(*edge: nodedge.edge.Edge*)
Add *Edge* to this *Scene*

**Parameters** **edge** – *Edge* to be added to this *Scene*

**Returns** *Edge*

**removeNode**(*nodeToRemove: nodedge.node.Node*)
Remove *Node* from this *Scene* :param nodeToRemove: *Node* to be removed from this *Scene* :type nodeToRemove: *Node*

**removeEdge**(*edgeToRemove: nodedge.edge.Edge*)
Remove *Edge* from this *Scene*

**Parameters** **edgeToRemove** – *Edge* to be remove from this *Scene*

**Returns** *Edge*

**clear**()
Remove all *Nodes* from this *Scene*. This causes also to remove all *Edges*

**saveToFile**(*filename*)
Save this *Scene* to the file on disk.

**Parameters** **filename** (str) – where to save this scene

**loadFromFile**(*filename*)
Load *Scene* from a file on disk

**Parameters** **filename** (str) – from what file to load the *Scene*

**Raises** *InvalidFile* if there was an error decoding JSON file.

**serialize**() → collections.OrderedDict
Serialization method to serialize this class data into OrderedDict which can be stored in memory or file easily.

**Returns** data serialized in OrderedDict

**Return type** OrderedDict

**deserialize**(*data: dict*, *hashmap: Optional[dict] = None*, *restoreId: bool = True*) → bool
Deserialization method which take data in python dict format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (dict) – dictionary containing serialized data
- **hashmap** (dict) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (bool) – True if we are creating new sockets. False is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** True if deserialization was successful, False otherwise

**Return type** bool

**getNodeClassFromData**(*data*)
Takes *Node* serialized data and determines which *Node Class* to instantiate according the description in the serialized Node.

**Parameters** **data** (dict) – serialized *Node* object data

**Returns** Instance of *Node* class to be used in this Scene

**Return type** *Node* class instance

**setNodeClassSelector**(*classSelectingFunction*)
Set the function which decides what *Node* class to instantiate during *Scene* deserialization. If not set, we will always instantiate *Node* for each *Node* in the *Scene*

**Parameters** **classSelectingFunction** (function) – function which returns *Node* class type (not instance) from *Node* serialized dict data

**Returns** Class Type of *Node* to be instantiated during deserialization

**Return type** *Node* class type

**itemAt**(*pos*)
Shortcut for retrieving item at provided *Scene* position

**Parameters** **pos** (QPointF) – scene position

**Returns** Qt Graphics Item at scene position

**Return type** QGraphicsItem

**exception** nodedge.scene.**InvalidFile**
Bases: Exception

### 3.2.17 nodedge.scene_clipboard

Scene clipboard module containing *SceneClipboard*.

**class** nodedge.scene_clipboard.**SceneClipboard**(*scene: Scene*)
Bases: object

*SceneClipboard* class

The scene clipboard class contains the code for the serialization/deserialization from/to the clipboard.

**serializeSelected**(*delete=False*)
Serializes selected items in the scene into OrderedDict.

**Parameters** **delete** (bool) – True to delete selected items after serialization. It is useful for cut operations.

**Returns** serialized data of current selection in Nodedge's *Scene*

**deserialize**(*data*)
Deserializes data from the clipboard.

**Parameters** **data** (dict) – dict data for deserialization to the Nodedge.node_scene.Scene

### 3.2.18 nodedge.scene_history

Scene history module containing *SceneHistory* class.

**class** nodedge.scene_history.**SceneHistory**(*scene: Scene*, *maxLength: int = 32*)
Bases: object

*SceneHistory* class

It contains the code for storing all the previous actions of the user in a dictionary.

**Parameters** **scene** (*Scene*) – reference to the *Scene*

**currentStep**

**stackSize**

**stack**

**addHistoryModifiedListener**(*callback*)
Register the callback associated with a *HistoryModified* event.

**Parameters** **callback** – callback function

**addHistoryStoredListener**(*callback*)
Register the callback associated with a *HistoryStored* event.

**Parameters** **callback** – callback function

**addHistoryRestoredListener**(*callback*)
Register the callback associated with a *HistoryRestored* event.

**Parameters** **callback** – callback function

**clear**(*storeInitialStamp=True*)
Reset the history stack.

**storeInitialStamp**()
Helper function usually used when new or open file operations are requested.

**canUndo**
This property returns `True` if the undo operation is available for the current history stack.

**Return type** `bool`

**canRedo**
This property returns `True` if the redo operation is available for the current history stack.

**Return type** `bool`

**undo**()
Perform the undo operation.

**redo**()
Perform the redo operation

**store**(*desc*, *sceneIsModified=True*)
Store the history stamp into the history stack.

**Parameters**

- **desc** (`str`) – Description of current history stamp
- **sceneIsModified** (`bool`) – if `True` marks that *Scene* has been modified.

Triggers:

- *History Modified*
- *History Stored*

**restore**()
Restore history stamp from history stack.

Triggers:

- *History Modified* event
- *History Restored* event

**restoreStamp**(*stamp*)
Restore history stamp to the current scene, included indication of the selected items.

**Parameters** **stamp** (dict) – history stamp to restore

**restoreStep**(*step*)

### 3.2.19 nodedge.serializable

Serializable "interface" module. It is an abstract class.

**class** nodedge.serializable.**Serializable**
Bases: object

Create data which are common to any serializable object.

It stores the id of the object used in the *SceneHistory*, the *SceneClipboard*, and the file structure.

**serialize**() → collections.OrderedDict
Serialization method to serialize this class data into OrderedDict which can be stored in memory or file easily.

**Returns** data serialized in OrderedDict

**Return type** OrderedDict

**deserialize**(*data: dict*, *hashmap: Optional[dict] = None*, *restoreId: bool = True*) → bool
Deserialization method which take data in python dict format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (dict) – dictionary containing serialized data
- **hashmap** (dict) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (bool) – True if we are creating new sockets. False is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** True if deserialization was successful, False otherwise

**Return type** bool

### 3.2.20 nodedge.socket

Socket module containing Nodedge's class for representing *Socket* class and *SocketLocation* constants.

**class** nodedge.socket.**SocketLocation**
Bases: enum.IntEnum

An enumeration.

**LEFT_TOP = 1**
Left top

**LEFT_CENTER = 2**
Left center

**LEFT_BOTTOM = 3**
Left bottom

**RIGHT_TOP = 4**
Right top

**RIGHT_CENTER = 5**
Right center

**RIGHT_BOTTOM = 6**
Right bottom

**class** nodedge.socket.**Socket**(*node: Node*, *index: int = 0*, *location: int = <SocketLocation.LEFT_TOP: 1>*, *socketType: int = 1*, *allowMultiEdges: bool = True*, *countOnThisNodeSide: int = 1*, *isInput: bool = False*)
Bases: *nodedge.serializable.Serializable*

Class representing input/output sockets of the nodes.

**Parameters**

- **node** (*Node*) – reference to the *Node* containing this socket
- **index** (int) – current index of this socket in the position
- **location** (*SocketLocation*) – socket position
- **socketType** – constant defining type of this socket. Every type is visually associated to a color.
- **allowMultiEdges** (bool) – attribute that defines if this socket can have multiple connected edges
- **countOnThisNodeSide** (int) – number of total sockets on this socket side, i.e. input/output
- **isInput** (bool) – attribute that defines whether this is an input or an output socket

**GraphicsSocketClass**
alias of *nodedge.graphics_socket.GraphicsSocket*

**socketType**

**isOutput**

**Getter** Return *True* is the *Socket* is not an input, *False* otherwise.

**Return type** bool

**delete**()
Delete this *Socket* from *Scene*.

**hasAnyEdge**
Whether the *Socket* has any *Edge* connected to it.

**Returns** True if any *Edge* is connected to this *Socket*

**Return type** bool

**pos**

**Returns** return this socket's position according the implementation stored in *Node*

**Return type** x, y position

**isConnected**(*edge: Edge*) → bool
Returns True if *Edge* is connected to this *Socket*.

**Parameters** **edge** (*Edge*) – *Edge* to check if it is connected to this *Socket*

**Returns** True if *Edge* is connected to this socket

**Return type** bool

**updateSocketPos**() → None
Helper function to set the graphical socket position. The exact socket position is calculated inside *Node*.

**addEdge**(*edge: Optional[Edge] = None*) → None
Append an *Edge* to the list of the connected *Edge*.

**Parameters** **edge** (*Edge*) – *Edge* to connect to this *Socket*

**removeEdge**(*edgeToRemove: Edge*) → None
Disconnect passed *Edge* from this *Socket*.

**Parameters** **edgeToRemove** (*Edge*) – *Edge* to disconnect

**removeAllEdges**(*silent=False*) → None
Disconnect all *Edge* from this *Socket*.

**Parameters** **silent** (bool) – If true, remove the edge without notifications

**static determineAllowMultiEdges**(*data*)
Deserialization helper function.

**Note:** This function is here to help solve the issue of opening older files in the newer format.

If the *allowMultiEdges* param is missing in the dictionary, we determine if this *Socket* should support multiple *Edge*.

**Parameters** **data** (dict) – socket's data in dict format for deserialization

**Returns** True if this socket should support multiple edges

**serialize**() → collections.OrderedDict
Serialization method to serialize this class data into OrderedDict which can be stored in memory or file easily.

**Returns** data serialized in OrderedDict

**Return type** OrderedDict

**deserialize**(*data: dict*, *hashmap: Optional[dict] = None*, *restoreId: bool = True*) → bool
Deserialization method which take data in python dict format with helping *hashmap* containing references to existing entities.

**Parameters**

- **data** (dict) – dictionary containing serialized data
- **hashmap** (dict) – helper dictionary containing references (by id == key) to existing objects
- **restoreId** (bool) – True if we are creating new sockets. False is useful when loading existing sockets which we want to keep the existing object's *id*

**Returns** True if deserialization was successful, False otherwise

**Return type** bool

### 3.2.21 nodedge.utils

Utils module with some helper functions.

nodedge.utils.**dumpException**(*e=None*, *file=None*)
Print out an exception message with the traceback to the console.

**Parameters**

- **e** (*Exception*) – Exception to print out
- **file** (str) – optional, file where the exception is dumped

nodedge.utils.**loadStyleSheet**(*fileName*)
Load an qss stylesheet to current QApplication instance.

**Parameters** **fileName** (str) – filename of qss stylesheet

nodedge.utils.**loadStyleSheets**(**args*)
Load multiple qss stylesheets. It concatenates them together and applies the final stylesheet to current QApplication instance.

**Parameters** **args** (str, str,...) – variable number of filenames of qss stylesheets

## 3.3 Module contents

CHAPTER 4

# Examples

The source code for the examples below are available in the *examples/* subdirectory of the source code distribution. The can also be accessed online via the [nodedge GitHub repository](https://github.com/nodedge/nodedge/tree/master/examples).

## 4.1 Python scripts

CHAPTER 5

# Evaluation

TL;DR: The evaluation system uses *eval()* and *evalChildren()*. `eval` is supposed to be overridden by your own implementation. The evaluation logic uses Flags for marking the *Nodes* to be *Dirty* and/or *Invalid*.

## 5.1 Evaluation Functions

There are 2 main methods used for evaluation:

- *eval()*
- *evalChildren()*

These functions are mutually exclusive. That means that `evalChildren` does **not** eval current *Node*, but only children of the current *Node*.

By default the implementation of *eval()* is "empty" and return 0. However it seems logical, that eval (if successfull) resets the *Node* not to be *Dirty* nor *Invalid*. This method is supposed to be overridden by your own implementation. As an example, you can check out the repository's `examples/example_calculator` to have an inspiration how to setup the *Node* evaluation on your own.

The evaluation takes advantage of *Node* flags described below.

## 5.2 `Node` Flags

Each *Node* has 2 flags:

- `Dirty`
- `Invalid`

The *Invalid* flag has always higher priority. That means when the *Node* is *Invalid* it doesn't matter if it is *Dirty* or not.

To mark a node *Dirty* or *Invalid* there are respective methods `markDirty()` and `markInvalid()`. Both methods take *bool* parameter for the new state. You can mark *Node* dirty by setting the parameter to `True`. Also you can un-mark the state by passing `False` value.

For both flags there are 3 methods available:

- `markInvalid()` - to mark only the *Node*
- *markChildrenInvalid()* - to mark only the direct (first level) children of the *Node*
- *markDescendantsInvalid()* - to mark it self and all descendant children of the *Node*

The same goes for the *Dirty* flag of course:

- `markDirty()` - to mark only the *Node*
- *markChildrenDirty()* - to mark only the direct (first level) children of the *Node*
- *markDescendantsDirty()* - to mark it self and all descendant children of the *Node*

Descendants or Children are always connected to Output(s) of current *Node*.

When a node is marked *Dirty* or *Invalid* event methods *onMarkedInvalid() onMarkedDirty()* are being called. By default, these methods do nothing. But still they are implemented in case you would like to override them and use in you own evaluation system.

CHAPTER 6

# Event system

Nodedge uses its own events (and tries to avoid using `pyqtSignal`) to handle logic happening inside the Scene. If a class does handle some events, they are usually described at the top of the page in this documentation.

Any of the events is subscribable to and the methods for registering callback are called:

```
add<EventName>Listener(callback)
```

You can register to any of these events any time.

## 6.1 Events used in Nodedge:

### 6.1.1 `Scene`

***Has Been Modified*** when something has changed in the *Scene*

***Item Selected*** when *Node* or *Edge* is selected

***Items Deselected*** when deselect everything appears

***Drag Enter*** when something is Dragged onto the *Scene*. Here we do allow or deny the drag

***Drop*** when we Drop something into the *Scene*

### 6.1.2 `SceneHistory`

***History Modified*** after *History Stamp* has been stored or restored

***History Stored*** after *History Stamp* has been stored

***History Restored*** after *History Stamp* has been restored

CHAPTER 7

# Serialization

All of serializable classes derive from *Serializable* class. *Serializable* does create commonly used parameters for our classes. In our case it is just `id` attribute.

*Serializable* defines two methods which should be overridden in child classes:

- *serialize()*
- *deserialize()*

According to coding-standards we keep these two functions on the bottom of the class source code.

To contain all of the data we use `OrderedDict` instead of regular *dict*. Mainly because we want to retain the order of parameters serialized in files.

Classes which derive from `Serializable`:

- *Scene*
- *Node*
- `QDMNodeContentWidget`
- *Edge*
- *Socket*
- genindex

### Development

You can check out the latest version of the source code with the command:

```
git clone https://github.com/nodedge/nodedge.git
```

You can run a set of unit tests to make sure that everything is working correctly. After installation, run:

```
python setup.py test
```

Your contributions are welcome! Simply fork the GitHub repository and send a pull request.

### Links

- Issue tracker: https://github.com/nodedge/nodedge/issues

# Python Module Index

## n

# Index

## A



## B



## C

## D



## E

## F



## G



## H

## I

## O

## P



## Q



## R



## S

## T



## U



## V



## W