Class Node

```
object --+
         |
        nuke.Node
```

Known Subclasses:
Group, Viewer

Instance Methods

[hide private]

Class of node	`Class(self)` Returns: Class of node.
	`__getitem__(x, y)` x[y]
	`__len__(x)` len(x)
a new object with type S, a subtype of T	`__new__(T, S, ...)`
	`__reduce_ex__(...)` helper for pickle
	`__repr__(x)` repr(x)
	`__str__(x)` str(x)
None	`addKnob(self, k)` Add knob k to this node or panel.
dict	`allKnobs(self)` Get a dictionary of (name, knob) pairs for all knobs in this node.
None	`autoplace(self)` Automatically place nodes, so they do not overlap.
List of x, y, w, h	`bbox(self)` Bounding box of the node.
bool	`canSetInput(self, i, node)` Check whether the output of 'node' can be connected to input i.
String list	`channels(self)` List channels output by this node.
Number of clones	`clones(self)` Returns: Number of clones.
bool	`connectInput(self, i, node)` Connect the output of 'node' to the i'th input or the next available unconnected input.
Floating point value	`deepSample(self, c, x, y, n)` Return pixel values from a deep image.
Integer value	`deepSampleCount(self, x, y)` Return number of samples for a pixel on a deep image.
List of nodes	`dependencies(self, what)` List all nodes referred to by this node.
List of nodes	`dependent(self, what)` List all nodes that read information from this node.
bool	`error()` True if the node or any in its input tree have an error, or False otherwise.
int	`firstFrame(self)` First frame in frame range for this node.
None	`forceValidate(self)` Force the node to validate itself, updating its hash.
Format	`format(self)` Format of the node.
FrameRange	`frameRange(self)` Frame range for this node.
str	`fullName(self)` Get the name of this node and any groups enclosing it in 'group.group.name' form.
The number of knobs	`getNumKnobs(self)` Returns: The number of knobs.
bool	`hasError()` True if the node itself has an error, regardless of the state of the ops in its input tree, or False otherwise.
int	`height(self)` Height of the node.
str	`help(self)` Returns: Help for the node.
None	`hideControlPanel(self)` Returns: None
The i'th input	`input(self, i)`

	Returns: The i'th input.
Number of inputs	**inputs**(self) Returns: Number of inputs.
bool	**isSelected**(self) Returns the current selection state of the node.
The knob named p or the pth knob	**knob**(self, p) Returns: The knob named p or the pth knob.
dict	**knobs**(self) Get a dictionary of (name, knob) pairs for all knobs in this node.
int	**lastFrame**(self) Last frame in frame range for this node.
List	**linkableKnobs**(self, knobType) Returns a list of any knobs that may be linked to from the node as well as some meta information about the knob.
Maximum number of inputs this node can have	**maxInputs**(self) Returns: Maximum number of inputs this node can have.
Maximum number of outputs this node can have	**maxOutputs**(self) Returns: Maximum number of outputs this node can have.
Maximum number of inputs this node can have	**maximumInputs**(self) Returns: Maximum number of inputs this node can have.
Maximum number of outputs this node can have	**maximumOutputs**(self) Returns: Maximum number of outputs this node can have.
value or dict	**metadata**(self, key, time, view) Return the metadata item for key on this node at current output context, or at optional time and view.
Minimum number of inputs this node can have	**minInputs**(self) Returns: Minimum number of inputs this node can have.
Minimum number of inputs this node can have	**minimumInputs**(self) Returns: Minimum number of inputs this node can have.
str	**name**(self) Returns: Name of node.
The number of knobs	**numKnobs**(self) Returns: The number of knobs.
list of int	**opHashes**(self) Returns a list of hash values, one for each op in this node.
Number of first optional input	**optionalInput**(self) Returns: Number of first optional input.
int	**pixelAspect**(self) Pixel Aspect ratio of the node.
bool	**proxy**(self) Returns: True if proxy is enabled, False otherwise.
None	**readKnobs**(self, s) Read the knobs from a string (TCL syntax).
None	**redraw**(self) Force a redraw of the node.
None	**removeKnob**(self, k) Remove knob k from this node or panel.
None	**resetKnobsToDefault**(self) Reset all the knobs to their default values.
Node rendering when paralled threads are running or None	**running**(self) Class method.
Floating point value	**sample**(self, c, x, y, dx, dy) Return pixel values from an image.
int	**screenHeight**(self) Height of the node when displayed on screen in the DAG, at 1:1 zoom, in pixels.
int	**screenWidth**(self) Width of the node when displayed on screen in the DAG, at 1:1 zoom, in pixels.
bool	**setInput**(self, i, node) Connect input i to node if canSetInput() returns true.
None	**setName**(self, name, uncollide=True) Set name of the node and resolve name collisions if optional named argument 'uncollide' is True.
None	**setSelected**(self, selected) Set the selection state of the node.
None	**setXYpos**(self, x, y) Set the (x, y) position of node in node graph.
None	**setXpos**(self, x) Set the x position of node in node graph.

None	`setYpos(self, y)` Set the y position of node in node graph.
None	`showControlPanel(self, forceFloat= false)` Returns: None
None	`showInfo(self, s)` Creates a dialog box showing the result of script s.
true if the properties panel is open	`shown(self)` This can be used to skip updates that are not visible to the user.
bool	`treeHasError()` True if the node or any in its input tree have an error, or False otherwise.
int	`width(self)` Width of the node.
String in .nk form	`writeKnobs(self, i)` Return a tcl list.
X position of node in node graph	`xpos(self)` Returns: X position of node in node graph.
Y position of node in node graph	`ypos(self)` Returns: Y position of node in node graph.
Inherited from `object`: `__delattr__`, `__format__`, `__getattribute__`, `__hash__`, `__init__`, `__reduce__`, `__setattr__`, `__sizeof__`, `__subclasshook__`	

Properties

[hide private]

Inherited from `object`: `__class__`

Method Details

[hide private]

Class(self)

Returns: Class of node
Class of node.

__new__(T, S, ...)

Returns: a new object with type S, a subtype of T
Overrides: object.__new__

__reduce_ex__(...)

helper for pickle

Overrides: object.__reduce_ex__
(inherited documentation)

__repr__(x)
(Representation operator)

repr(x)

Overrides: object.__repr__

__str__(x)
(Informal representation operator)

str(x)

Overrides: object.__str__

addKnob(self, k)

Add knob k to this node or panel.

Parameters:
- **`k`** - Knob.

Returns: None
None.

allKnobs(self)

Get a dictionary of (name, knob) pairs for all knobs in this node.

For example:

```
>>> b = nuke.nodes.Blur()
>>> b.knobs()
```

Returns: dict

Dictionary of all knobs.

***autoplace*(*self*)**

Automatically place nodes, so they do not overlap.

Returns: None
None.

***bbox*(*self*)**

Bounding box of the node.

Returns: List of x, y, w, h
List of x, y, w, h.

***canSetInput*(*self*, *i*, *node*)**

Check whether the output of 'node' can be connected to input i.

Parameters:

- **`i`** - Input number.
- **`node`** - The node to be connected to input i.

Returns: bool
True if node can be connected, False otherwise.

***channels*(*self*)**

List channels output by this node.

Returns: String list
String list.

***clones*(*self*)**

Returns: Number of clones
Number of clones.

***connectInput*(*self*, *i*, *node*)**

Connect the output of 'node' to the i'th input or the next available unconnected input. The requested input is tried first, but if it is already set then subsequent inputs are tried until an unconnected one is found, as when you drop a connection arrow onto a node in the GUI.

Parameters:

- **`i`** - Input number to try first.
- **`node`** - The node to connect to input i.

Returns: bool
True if a connection is made, False otherwise.

***deepSample*(*self*, *c*, *x*, *y*, *n*)**

Return pixel values from a deep image. This requires the image to be calculated, so performance may be very bad if this is placed into an expression in a control panel.

Parameters:

- **`c`** - Channel name.
- **`x`** - Position to sample (X coordinate).
- **`y`** - Position to sample (Y coordinate).
- **`n`** - Sample index (between 0 and the number returned by deepSampleCount() for this pixel, or -1 for the frontmost).

Returns: Floating point value
Floating point value.

***deepSampleCount*(*self*, *x*, *y*)**

Return number of samples for a pixel on a deep image. This requires the image to be calculated, so performance may be very bad if this is placed into an expression in a control panel.

Parameters:

- **`x`** - Position to sample (X coordinate).
- **`y`** - Position to sample (Y coordinate).

Returns: Integer value
Integer value.

***dependencies*(*self*, *what*)**

```
List all nodes referred to by this node. 'what' is an optional integer (see below).
You can use the following constants or'ed together to select what types of dependencies are looked for:
         nuke.EXPRESSIONS = expressions
         nuke.INPUTS = visible input pipes
         nuke.HIDDEN_INPUTS = hidden input pipes.
The default is to look for all types of connections.

Example:
nuke.toNode('Blur1').dependencies( nuke.INPUTS | nuke.EXPRESSIONS )
@param what: Or'ed constant of nuke.EXPRESSIONS, nuke.INPUTS and nuke.HIDDEN_INPUTS to select the types of dependencies. The default is to lo
@return: List of nodes.
```

Returns: List of nodes

dependent(self, what)

```
List all nodes that read information from this node.  'what' is an optional integer (see below).
You can use any combination of the following constants or'ed together to select what types of dependent nodes to look for:
         nuke.EXPRESSIONS = expressions
         nuke.INPUTS = visible input pipes
         nuke.HIDDEN_INPUTS = hidden input pipes.
The default is to look for all types of connections.

Example:
nuke.toNode('Blur1').dependent( nuke.INPUTS | nuke.EXPRESSIONS )
@param what: Or'ed constant of nuke.EXPRESSIONS, nuke.INPUTS and nuke.HIDDEN_INPUTS to select the types of dependent nodes. The default is to
@return: List of nodes.
```

Returns: List of nodes

error()

True if the node or any in its input tree have an error, or False otherwise.

Error state of the node and its input tree. Deprecated; use hasError or treeHasError instead. Note that this will always return false for viewers, which cannot generate their input trees. Instead, choose an input of the viewer (e.g. the active one), and call treeHasError() on that.

Returns: bool

firstFrame(self)

First frame in frame range for this node.

Returns: int
int.

format(self)

Format of the node.

Returns: Format
Format.

frameRange(self)

Frame range for this node.

Returns: FrameRange
FrameRange.

fullName(self)

Get the name of this node and any groups enclosing it in 'group.group.name' form.

Returns: str
The fully-qualified name of this node, as a string.

getNumKnobs(self)

Returns: The number of knobs
The number of knobs.

hasError()

True if the node itself has an error, regardless of the state of the ops in its input tree, or False otherwise.

Error state of the node itself, regardless of the state of the ops in its input tree. Note that an error on a node may not appear if there is an error somewhere in its input tree, because it may not be possible to validate the node itself correctly in that case.

Returns: bool

height(self)

Height of the node.

Returns: int
    int.

help(self)

Returns: str
    Help for the node.

hideControlPanel(self)

Returns: None
    None

input(self, i)

Parameters:

- **`i`** - Input number.

Returns: The i'th input
    The i'th input.

inputs(self)

Returns: Number of inputs
    Number of inputs.

isSelected(self)

Returns the current selection state of the node. This is the same as checking the 'selected' knob.

Returns: bool
    True if selected, or False if not.

knob(self, p)

Parameters:

- **`p`** - A string or an integer.

Returns: The knob named p or the pth knob
    The knob named p or the pth knob.

knobs(self)

Get a dictionary of (name, knob) pairs for all knobs in this node.

For example:

```
>>> b = nuke.nodes.Blur()
>>> b.knobs()
```

Returns: dict
    Dictionary of all knobs.

lastFrame(self)

Last frame in frame range for this node.

Returns: int
    int.

linkableKnobs(self, knobType)

Returns a list of any knobs that may be linked to from the node as well as some meta information about the knob. This may include whether the knob is enabled and whether it should be used for absolute or relative values. Not all of these variables may make sense for all knobs..

Parameters:

- **`knobType`**, **`A`**, **`KnobType`**, **`describing`**, **`the`**, **`type`**, **`of`**, **`knobs`**, **`you`**, **`want.@return`** - A list of LinkableKnobInfo that may be empty .

Returns: List

maxInputs(self)

Returns: Maximum number of inputs this node can have
Maximum number of inputs this node can have.

maxOutputs(self)

Returns: Maximum number of outputs this node can have
Maximum number of outputs this node can have.

maximumInputs(self)

Returns: Maximum number of inputs this node can have
Maximum number of inputs this node can have.

maximumOutputs(self)

Returns: Maximum number of outputs this node can have
Maximum number of outputs this node can have.

metadata(self, key, time, view)

Return the metadata item for key on this node at current output context, or at optional time and view. If key is not specified a dictionary containing all key/value pairs is returned. None is returned if key does not exist on this node.

Parameters:
- `key` - Optional name of the metadata key to retrieve.
- `time` - Optional time to evaluate at (default is taken from node's current output context).
- `view` - Optional view to evaluate at (default is taken from node's current output context).

Returns: value or dict
The requested metadata value, a dictionary containing all keys if a key name is not provided, or None if the specified key is not matched.

minInputs(self)

Returns: Minimum number of inputs this node can have
Minimum number of inputs this node can have.

minimumInputs(self)

Returns: Minimum number of inputs this node can have
Minimum number of inputs this node can have.

name(self)

Returns: str
Name of node.

numKnobs(self)

Returns: The number of knobs
The number of knobs.

optionalInput(self)

Returns: Number of first optional input
Number of first optional input.

pixelAspect(self)

Pixel Aspect ratio of the node.

Returns: int
float.

proxy(self)

Returns: bool
True if proxy is enabled, False otherwise.

readKnobs(self, s)

Read the knobs from a string (TCL syntax).

Parameters:

- **`s`** - A string.

Returns: None
None.

redraw(self)

Force a redraw of the node.

Returns: None
None.

removeKnob(self, k)

Remove knob k from this node or panel. Throws a ValueError exception if k is not found on the node.

Parameters:
- **`k`** - Knob.

Returns: None
None.

running(self)

Class method.

Returns: Node rendering when paralled threads are running or None
Node rendering when paralled threads are running or None.

sample(self, c, x, y, dx, dy)

Return pixel values from an image. This requires the image to be calculated, so performance may be very bad if this is placed into an expression in a control panel. Produces a cubic filtered result. Any sizes less than 1, including 0, produce the same filtered result, this is correct based on sampling theory. Note that integers are at the corners of pixels, to center on a pixel add .5 to both coordinates. If the optional dx,dy are not given then the exact value of the square pixel that x,y lands in is returned. This is also called 'impulse filtering'.

Parameters:
- **`c`** - Channel name.
- **`x`** - Centre of the area to sample (X coordinate).
- **`y`** - Centre of the area to sample (Y coordinate).
- **`dx`** - Optional size of the area to sample (X coordinate).
- **`dy`** - Optional size of the area to sample (Y coordinate).

Returns: Floating point value
Floating point value.

screenHeight(self)

Height of the node when displayed on screen in the DAG, at 1:1 zoom, in pixels.

Returns: int
int.

screenWidth(self)

Width of the node when displayed on screen in the DAG, at 1:1 zoom, in pixels.

Returns: int
int.

setInput(self, i, node)

Connect input i to node if canSetInput() returns true.

Parameters:
- **`i`** - Input number.
- **`node`** - The node to connect to input i.

Returns: bool
True if canSetInput() returns true, or if the input is already correct.

setName(self, name, uncollide=True)

Set name of the node and resolve name collisions if optional named argument 'uncollide' is True.

Parameters:
- **`name`** - A string.
- **`uncollide`** - Optional boolean to resolve name collisions. Defaults to True.

Returns: None

None

***setSelected*(*self*, *selected*)**

Set the selection state of the node. This is the same as changing the 'selected' knob.

Parameters:

- **`selected`** - New selection state - True or False.

Returns: None
None.

***setXYpos*(*self*, *x*, *y*)**

Set the (x, y) position of node in node graph.

Parameters:

- **`x`** - The x position of node in node graph.
- **`y`** - The y position of node in node graph.

Returns: None
None.

***setXpos*(*self*, *x*)**

Set the x position of node in node graph.

Parameters:

- **`x`** - The x position of node in node graph.

Returns: None
None.

***setYpos*(*self*, *y*)**

Set the y position of node in node graph.

Parameters:

- **`y`** - The y position of node in node graph.

Returns: None
None.

***showControlPanel*(*self*, *forceFloat*= *false*)**

Parameters:

- **`forceFloat`** - Optional python object. If it evaluates to True the control panel will always open as a floating panel. Default is False.

Returns: None
None

***showInfo*(*self*, *s*)**

Creates a dialog box showing the result of script s.

Parameters:

- **`s`** - A string.

Returns: None
None.

***shown*(*self*)**

This can be used to skip updates that are not visible to the user.

Returns: true if the properties panel is open
true if the properties panel is open. This can be used to skip updates that are not visible to the user.

***treeHasError*()**

True if the node or any in its input tree have an error, or False otherwise.

Error state of the node and its input tree. Note that this will always return false for viewers, which cannot generate their input trees. Instead, choose an input of the viewer (e.g. the active one), and call treeHasError() on that.

Returns: bool

***width*(*self*)**

Width of the node.

Returns: int
int.

writeKnobs(self, i)

```
Return a tcl list. If TO_SCRIPT | TO_VALUE is not on, this is a simple list
of knob names. If it is on, it is an alternating list of knob names
and the output of to_script().

Flags can be any of these or'd together:
- nuke.TO_SCRIPT produces to_script(0) values
- nuke.TO_VALUE produces to_script(context) values
- nuke.WRITE_NON_DEFAULT_ONLY skips knobs with not_default() false
- nuke.WRITE_USER_KNOB_DEFS writes addUserKnob commands for user knobs
- nuke.WRITE_ALL writes normally invisible knobs like name, xpos, ypos

@param i: The set of flags or'd together. Default is TO_SCRIPT | TO_VALUE.
@return: String in .nk form.
```

Returns: String in .nk form

xpos(self)

Returns: X position of node in node graph
X position of node in node graph.

ypos(self)

Returns: Y position of node in node graph
Y position of node in node graph.