```
class Node
{
public:
    Node() = default;
    virtual ~Node() = default;

    //==============================================================================
    /** Call once after the graph has been constructed to initialise buffers etc. */
    void initialise (const PlaybackInitialisationInfo&);

    /** Call before processing the next block, used to reset the process status. */
    void prepareForNextBlock (juce::Range<int64_t> referenceSampleRange);

    /** Call to process the node, which will in turn call the process method with the
        buffers to fill.
        @param referenceSampleRange The monotonic stream time in samples.
                                    This will be passed to the ProcessContext during the
                                    process callback so nodes can use this to determine file
                                    reading positions etc.
                                    Some nodes may ignore this completely but it should at the
                                    least specify the number to samples to process in this block.
    */
    void process (juce::Range<int64_t> referenceSampleRange);

    /** Returns true if this node has processed and its outputs can be retrieved. */
    bool hasProcessed() const;

    /** Contains the buffers for a processing operation. */
    struct AudioAndMidiBuffer
    {
        juce::dsp::AudioBlock<float> audio;
        tracktion_engine::MidiMessageArray& midi;
    };

    /** Returns the processed audio and MIDI output.
        Must only be called after hasProcessed returns true.
    */
    AudioAndMidiBuffer getProcessedOutput();

    //==============================================================================
    /** Called after construction to give the node a chance to modify its topology.
        This should return true if any changes were made to the topology as this
        indicates that the method may need to be called again after other nodes have
        had their toplogy changed.
    */
    virtual bool transform (Node& /*rootNode*/) { return false; }

    /** Should return all the inputs directly feeding in to this node. */
    virtual std::vector<Node*> getDirectInputNodes() { return {}; }

    /** Should return the properties of the node.
        This should not be called until after initialise.
    */
    virtual NodeProperties getNodeProperties() = 0;

    /** Should return true when this node is ready to be processed.
        This is usually when its input's output buffers are ready.
    */
    virtual bool isReadyToProcess() = 0;

    /** Struct to describe a single iteration of a process call. */
    struct ProcessContext
    {
        juce::Range<int64_t> referenceSampleRange;
        AudioAndMidiBuffer buffers;
    };

    //==============================================================================
    /** @internal */
    void* internal = nullptr;

protected:
    /** Called once before playback begins for each node.
        Use this to allocate buffers etc.
        This step can be used to modify the topology of the graph (i.e. add/remove nodes).
        However, if you do this, you must make sure to call initialise on them so they are
        fully prepared for processing.
    */
    virtual void prepareToPlay (const PlaybackInitialisationInfo&) {}

    /** Called before once on all Nodes before they are processed.
        This can be used to prefetch audio data or update mute statuses etc..
    */
    virtual void prefetchBlock (juce::Range<int64_t> /*referenceSampleRange*/) {}

    /** Called when the node is to be processed.
        This should add in to the buffers available making sure not to change their size at all.
    */
    virtual void process (const ProcessContext&) = 0;
```

Node Overview

Node Overview

```
protected:
    /** Called once before playback begins for each node.
        Use this to allocate buffers etc.
        This step can be used to modify the topology of the graph (i.e. add/remove nodes).
        However, if you do this, you must make sure to call initialise on them so they are
        fully prepared for processing.
    */
    virtual void prepareToPlay (const PlaybackInitialisationInfo&) {}

    /** Called before once on all Nodes before they are processed.
        This can be used to prefetch audio data or update mute statuses etc..
    */
    virtual void prefetchBlock (juce::Range<int64_t> /*referenceSampleRange*/) {}

    /** Called when the node is to be processed.
        This should add in to the buffers available making sure not to change their size at all.
    */
    virtual void process (const ProcessContext&) = 0;
```

Implementing a Node Summary