prepareToPlay

```cpp
namespace node_player_utils
{
    /** Prepares a specific Node to be played and returns all the Nodes. */
    static std::vector<Node*> prepareToPlay (Node* node, Node* oldNode,
                                             double sampleRate, int blockSize)
    {
        if (node == nullptr)
            return {};

        // First give the Nodes a chance to transform
        transformNodes (*node);

        // Then find all the nodes as it might have changed after initialisation
        auto orderedNodes = tracktion_graph::getNodes (*node, tracktion_graph::VertexOrdering::postordering);

        // Next, initialise all the nodes, this will call prepareToPlay on
        const PlaybackInitialisationInfo info { sampleRate, blockSize, *node, oldNode };

        for (auto node : orderedNodes)
            node->initialise (info);

        // Finally return the Nodes in playback order
        return orderedNodes;
    }
}
```

prepareToPlay

```
namespace node_player_utils
{
    /** Prepares a specific Node to be played and returns all the Nodes. */
    static std::vector<Node*> prepareToPlay (Node* node, Node* oldNode,
                                             double sampleRate, int blockSize)
    {
        if (node == nullptr)
            return {};

        // First give the Nodes a chance to transform
        transformNodes (*node);

        // Then find all the nodes as it might have changed after initialisation
        auto orderedNodes = tracktion_graph::getNodes (*node, tracktion_graph::VertexOrdering::postordering);

        // Next, initialise all the nodes, this will call prepareToPlay on
        const PlaybackInitialisationInfo info { sampleRate, blockSize, *node, oldNode };

        for (auto node : orderedNodes)
            node->initialise (info);

        // Finally return the Nodes in playback order
        return orderedNodes;
    }
}
```

SimpleNodePlayer::Process

```
void process (const Node::ProcessContext& pc)
{
    // Prepare all nodes for the next block
    for (auto node : orderedNodes)
        node->prepareForNextBlock (pc.referenceSampleRange);

    // Then process them all in sequence
    for (auto node : orderedNodes)
        node->process (pc.referenceSampleRange);

    // Finally copy the output from the root Node to our player buffers
    auto output = rootNode->getProcessedOutput();
    const size_t numAudioChannels = std::min (output.audio.getNumChannels(),
                                              pc.buffers.audio.getNumChannels());

    if (numAudioChannels > 0)
        pc.buffers.audio.getSubsetChannelBlock (0, numAudioChannels)
                        .add (output.audio.getSubsetChannelBlock (0, numAudioChannels));

    pc.buffers.midi.mergeFrom (output.midi);
}
```