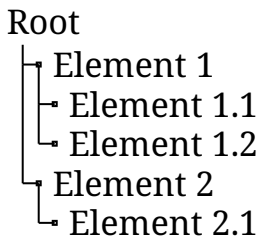
How to Render a (Hierarchical) Tree in Asciidoctor

Showing a hierarchical tree, like a file system directory tree, in Asciidoctor is surprisingly hard. We use PlantUML to render the tree on all common platforms.



Problem Description

Describe a hierarchical tree in an Asciidoctor document and render it as in a tree view.

Existing Approaches

We know of two Asciidoctor extensions tackling this problem:

- TreeBlockMacro Shows an actual file system tree. Available for Ruby Asciidoctor.
- Monotree Renders a structural description of a tree as AsciiArt. Available for AsciidoctorJ.

These approaches have some drawbacks:

- Only available for one platform
- Limited to file system trees, or AsciiArt

Solution: Creole Markup via PlantUML

Most platforms support the Asciidoctor Diagram extension, Which, in turn, supports PlantUML. Within PlantUML, we can use Creole to render a hierarchical tree.

We need a container to host the Creole markup. A `legend` seems suitable.

For the actual tree, we draw the tree as AsciiArt.

NOTE	Make sure to indent each tree level with **exactly two spaces**.

We use PlantUML's `skinparam` feature for formatting the output similar to Asciidoctor.

If we embed the PlantUML diagram with parameters `format=svg` and `opts="inline"`:

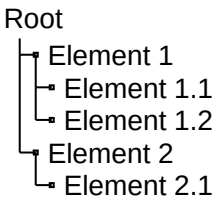
- The text stays as text (i.e. selectable) in both HTML and PDF output
- We don't need to ship additional files with the output

Examples

Simplest Variant: Only get rid of border and background

```
[plantuml, format=svg, opts="inline"]
----
skinparam Legend {
    BackgroundColor transparent
    BorderColor transparent
}
legend
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

Example 1. Simplest Variant: Only get rid of border and background

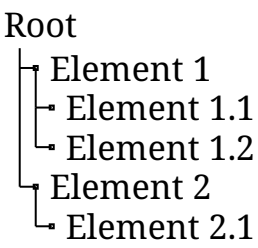


We might want to hide the different origin of the tree rendering by adjusting the font.

Use same font as default Asciidoctor style

```
[plantuml, format=svg, opts="inline"]
----
skinparam Legend {
    BackgroundColor transparent
    BorderColor transparent
    FontName "Noto Serif", "DejaVu Serif", serif
    FontSize 17
}
legend
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

Example 2. Use same font as default Asciidoctor style



We can use PlantUML's `!include` feature to externalize the formatting.

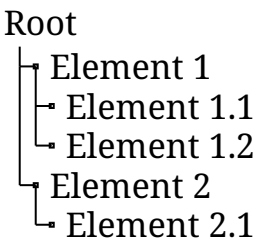
Externalized formatting

```
[plantuml, format=svg, opts="inline"]
----
!include asciidoctor-style.iuml
legend
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

asciidoctor-style.iuml

```
skinparam Legend {
    BackgroundColor transparent
    BorderColor transparent
    FontName "Noto Serif", "DejaVu Serif", serif
    FontSize 17
}
```

Example 3. Externalized formatting

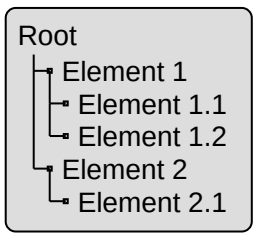


For reference, we show what the tree rendering looks like without any styling.

Barebone tree rendering

```
[plantuml, format=svg, opts="inline"]
----
legend
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

Example 4. Barebone tree rendering



Remove Dependency on Graphviz / Dot

IMPORTANT	Thanks to PlantUML's awesome response time, **trees won't depend on Graphviz any more from version 1.2019.11 onwards**.

PlantUML requires Graphviz to be installed. However, there's an exception: Rendering sequence diagrams does not depend on Graphviz. So we need to convince PlantUML to draw a sequence diagram *without actually drawing any diagram*.

This feels more like a hack, but the following works while leading to only slightly larger images.

Trick PlantUML into a sequence diagram to no longer have a depencency on Graphviz

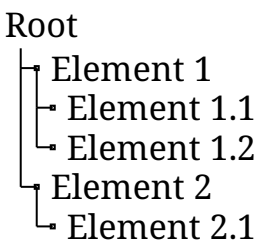
```
[plantuml, format=svg, opts="inline"]
----
skinparam Legend {
    BackgroundColor transparent
    BorderThickness 0
    FontName "Noto Serif", "DejaVu Serif", serif
    FontSize 17
}

' We use skinparams to hide our dummy participant
' and make it as little space as possible
skinparam SequenceLifeLineBorderThickness 0
skinparam SequenceLifeLineBorderColor transparent
skinparam SequenceParticipant {
    BackgroundColor transparent
    BorderColor transparent
    Shadowing false
    FontSize 0
    BorderThickness 0
    Padding 0
}
hide footbox

' "participant" nudges PlantUML to a sequence diagram
participant dummy

legend top left
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

Example 5. Trick PlantUML into a sequence diagram to no longer have a depencency on Graphviz



As this hack is quite ugly, we can hide it in an `!include`. The external file is only required at rendering time.

Hide PlantUML sequence diagram hack in external file

```
[plantuml, format=svg, opts="inline"]
----
!include nodot-asciidoctor-style.iuml
legend
Root
|_ Element 1
  |_ Element 1.1
  |_ Element 1.2
|_ Element 2
  |_ Element 2.1
end legend
----
```

nodot-asciidoctor-style.iuml

```
skinparam Legend {
    BackgroundColor transparent
    BorderColor transparent
    BorderThickness 0
    FontName "Noto Serif", "DejaVu Serif", serif
    FontSize 17
}
skinparam SequenceLifeLineBorderThickness 0
skinparam SequenceLifeLineBorderColor transparent
skinparam SequenceParticipant {
    BackgroundColor transparent
    BorderColor transparent
    Shadowing false
    FontSize 0
    BorderThickness 0
    Padding 0
}
hide footbox
participant dummy
```

Example 6. Hide PlantUML sequence diagram hack in external file