

Syntax Highlighting in Alan Documentation

Tristano Ajmone

Revision 2.3
March 8, 2019

Table of Contents

.....	ix
I. Syntaxes Preview	1
Alan	3
BNF	5
Game Transcripts	7
Shell	9
II. Syntaxes Details	11
Alan Syntax	13
Syntax Tests	13
Predefined Classes and Instances	13
Quoted Identifiers	14
Strings	15
False Positive Keywords Matches	15
Ligatures Support	15
Callouts Support	16
BNF Syntax	17
III. Customizing The Look of Code	19
Custom Color-Marking	21
Yellow Highlighting	21
Green Highlighting	21

List of Figures

1. Example of correct ligatures support in Sublime Text. 16

List of Examples

1. Game Transcript With Title 7

This document is for previewing how source code and other verbatim blocks will be rendered and styled in the final documentation, and for comparing quality and differences across all the supported output formats.

Some contents are format dependent and will vary according to the document's output format and the backend employed, providing additional information specific to that format and/or backend.



For syntax highlighting in the PDF backend we use [XSLTHL](#) for which we've created an Alan syntax definition from scratch.

Part I. Syntaxes Preview

The first part of this document offers a quick preview of all the colored syntaxes supported by the Alan Docs template.

In-depth information and contextual tests for syntaxes that support syntax highlighting can be found in [Part II](#) of this document.

Alan

Alan code examples are styled as `alan` source blocks.

```
-----
-- "Barracks Example" by Tristano Ajmone ❶
-----

The canteen IsA LOCATION ❷
  Name 'The Barrack''s Mess Hall'.
END The canteen.

The trolley IsA object At canteen
  Container Taking object.
  Limits
    Count 5 Else "The trolley is full!"
  Has Not been_examined.
  VERB examine
  DOES
    If This Has Not been_examined
      Then "This old trolley has surely seen better days. $nIt seems barely
        capable of carrying its own weight without collapsing, let alone
        endure its daily use!"
      Make This been_examined.
      Else "It's just an old trolley."
    End If.
  END VERB examine.
End The trolley.

-----

Start at canteen.
```

- ❶ Callouts use default colors, no customization.
- ❷ Callouts use default colors, no customization.

Alan Code With Title

```
Every book IsA object.
  Description "It's just a book."
End Every.
```



Currently, there aren't any Alan sourcecode examples with callouts in the *Alan Manual*.

BNF

BNF rules are styles as `bnf` source blocks.

```
class = 'EVERY' id
      [inheritance]
      {property}
      'END' 'EVERY' [id] ['.']
```

BNF Rules Block With Title

```
class = 'EVERY' id
      [inheritance]
      {property}
      'END' 'EVERY' [id] ['.']
```



Currently, there aren't any BNF rules with callouts in the *Alan Manual*.



CALLOUTS NOT SUPPORTED

Neither the HTML nor the PDF syntax highlighters are currently set to support BNF syntax. This means that callouts are not currently being processed and can't be used.

If callouts need to be used inside BNF rules, we need to either:

- Add a BNF syntax to the highlighters.
- Use a different type of verbatim block.

Game Transcripts

Game transcript use an `example` block with `role="gametranscript"`.

Although a game transcript is neither source code nor a verbatim block, I'm including it in this document to allow comparing its styles and colors to other block-type elements in the documentation.

West of House.

Welcome to Dungeon (ALAN Demo). This version created 29-FEB-92.

You are in an open field west of a big white house with a boarded front door. There is a small mailbox here.

> *examine the mailbox*

I see nothing special about the mailbox.

> *open it*

The small mailbox is now open. The small mailbox contains a leaflet.

Example 1. Game Transcript With Title

> *north*



CALLOUTS NOT SUPPORTED

Game transcripts are not verbatim blocks, so callouts are not supported.

Shell

For shell usage examples we use a `literal` block with `role="shell"`. The color scheme employed is based on the [new default console scheme of Windows 10](#).

```
D:\>alan -help ❶
Usage: ALAN <adventure> [-help] [options]

Arguments:
  <adventure>      -- file name, default extension '.alan' ❷ ❸

Options:
  -help            -- this help
  -[-]verbose      -- verbose messages (default: OFF)
  -[-]warnings     -- [don't] show warning messages (default: ON)
  -[-]infos        -- [don't] show informational messages (default: OFF)
  -include <path> -- additional directory to search after current when
                   looking for imported files (may be repeated)
```

- ❶ I've also styled the callouts background and foreground colors to match the shell scheme.
- ❷ By default, callouts have black circles and white numbers, which don't work well with the black background of the shell scheme.
- ❸ In the PDF document I haven't been able to tweak callouts colors.

Shell Sample With Title

```
D:\>dir
```



The *Alan Manual* contains shell examples with callouts in [App. F.1. Format of Messages](#).



CALLOUTS PROBLEM

Unlike the HTML backend, I haven't found a way to customize the background color of callout discs inside shell examples; therefore the default black colored reverse-video circles are lost in the black background of the shell color scheme.

The PDF template uses SVG image files to render the reverse-video circled numbers; although it's possible to customize the XSL settings to point to customized SVG images, this setting would affect all callouts — i.e. I haven't found a way to use different callout images depending on the block role.

Possible solutions to this problem:

- Provide customized SVG iamges with different background color (affects all callouts in the book).
- Disable callout icons and use text instead (i.e. (1)) via the `callout.graphics` setting.
- Use a lighter color scheme for shell (simplest solution).

Part II. Syntaxes Details

This part of the document covers in more detail the syntaxes which are syntax highlighted in the Alan Doc template, and provides also various tests to check how styles are rendered in various contexts.

Alan Syntax

Different conversion toolchains use different tools for syntax highlighting. For HTML format conversion, various highlighters are natively supported by Asciidoctor. For PDF conversion, the usable highlighting tool(s) will be dictated by the third party backend employed.

In all cases, a custom Alan syntax definition would have to be created for the highlighter tool used with each backend, as none of them ships with a native Alan definition (the sole exception being [Highlight](#), which is not natively supported in any of the backends, but could be integrated into some of them).



The PDF conversion toolchain uses [asciidoc-fopub](#) to create the PDF from the DocBook version of the documents. Asciidoc-fopub ships with an integrated syntax highlighter, called XSLHL.

Syntax Tests

Here follow some code snippets for testing both common- and edge-cases in the language and ensure they are rendered as expected.

Predefined Classes and Instances

Although it's usually unnecessary to highlight with separate styling the predefined Alan classes and instances (i.e. the `hero`), it might be desirable to be able to do so in tutorials and documentation, for educational purposes. For this reason, I've added a special highlighting group for the predefined classes, and another one for the `hero`. Both can be optionally enabled by assigning styles to them, or just be ignored (or set to the same style as normal text) to hide them in the highlighted source.

```
Synonyms me = hero.

The basement IsA location.
  Description "What a dark ang gloomy place!"
End The.

The vampire IsA actor At basement.
End The.

The coffin IsA object At basement.
  Container taking thing.
```

```
End The.

ADD TO EVERY thing
  VERB examine
  Does
    "You examine $+1, but find nothing unusual."
  END VERB.
END ADD TO.

ADD TO EVERY string
  VERB 'say'
  Does
    "You say ""$$" Say This. "$$""!"
  END VERB.
END ADD TO.

ADD TO EVERY integer
  VERB shout
  Does
    "You shout ""$$" Say This. "$$"" out loud!"
  END VERB.
END ADD TO.

ADD TO EVERY literal
  VERB whisper
  Does
    "You whisper ""$$" Say This. "$$""!"
  END VERB.
END ADD TO.

-----

Start at basement.
```

Quoted Identifiers

Although quoted identifiers shouldn't receive any special coloring in the highlighted code, the syntax definition must be made aware of them to prevent false-positive keywords matches for tokens inside quoted identifiers:

```
The 'At the Bus Stop' IsA location --> Watch out for: 'AT', 'THE' and 'STOP'!
End The.
```

This hidden feature will be implemented differently in each syntax definition, depending on how the specific syntax highlighting engine works.

However, in the final highlighted code, quoted identifiers are shown with the same color and style as the base (normal) text, effectively hiding to the end user the

difference between quoted identifiers and plain code (normal identifiers are just treated as plain code).

Single Quotes Escaping

To prevent breaking up syntax highlighting, quoted identifiers need to support correctly escaping single quotes inside them via `' '`. Although this won't be visible in the highlighted output of the following example, examining the document's source (with formats which support it, like HTML) would allow to check that the quoted identifier was correctly parsed and tagged by the highlighter:

```
The 'Bob''s House' IsA location --> Escaped apostrophe in location name.  
End The.
```

Strings

False Positive Keywords Matches

The following snippet tests against false positive keywords in strings and quoted identifiers. This should never happen, and if it does then the Alan syntax definition for that highlighter is flawed and needs to be fixed.

```
-- Tokens in strings and quoted IDs shouldn't be highlighted as keywords...  
  
The 'At the Bus Stop' IsA location --> Watch out for: 'AT', 'THE' and 'STOP'!  
  Description  
    "An old man waits for the bus here." --> 'AN', 'FOR', 'THE', 'HERE'!  
End The.
```

Ligatures Support

The following snippets is meant to test if the current backend and/or font supports ligatures. Some code fonts support ligatures replacements for common programming symbols, but they might differ in the way they represent `>=`, `<=`, `=>` and `=<`, depending on which combinations are assigned as comparison operators and which as arrows.

We must ensure that, if ligatures are supported in any converted document, Alan's `>=` (greater or equal), `<=` (less or equal) and `=>` (alternative Then in Rules) are correctly represented.

Here's the test snippet:

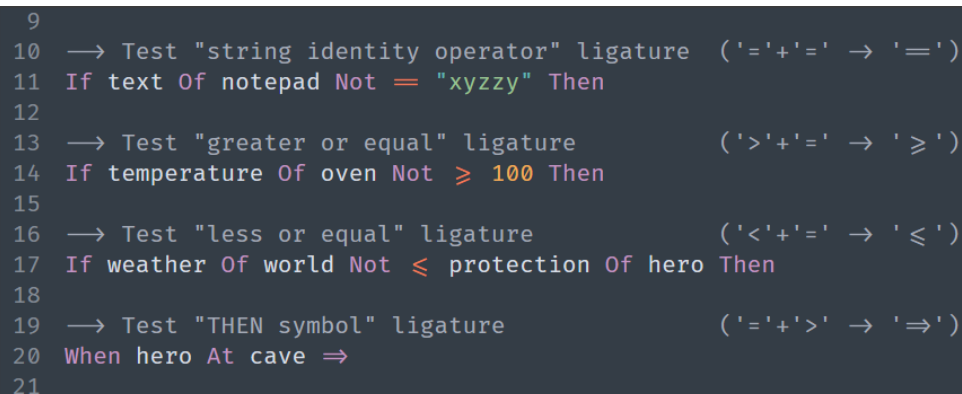
```
--> Test "string identity operator" ligature ('='+'=' -> '==')
If text Of notepad Not == "xyzyz" Then

--> Test "greater or equal" ligature ('>'+'=' -> '>=')
If temperature Of oven Not >= 100 Then

--> Test "less or equal" ligature ('<'+'=' -> '<=')
If weather Of world Not <= protection Of hero Then

--> Test "THEN symbol" ligature ('='+'>' -> '=>')
When hero At cave =>
```

And here's a screenshot of the above code displayed in Sublime Text 3 via the [Sublime Alan](#) package:



```
9
10 → Test "string identity operator" ligature ('='+'=' → '==')
11 If text Of notepad Not = "xyzyz" Then
12
13 → Test "greater or equal" ligature ('>'+'=' → '>=')
14 If temperature Of oven Not ≥ 100 Then
15
16 → Test "less or equal" ligature ('<'+'=' → '<=')
17 If weather Of world Not ≤ protection Of hero Then
18
19 → Test "THEN symbol" ligature ('='+'>' → '=>')
20 When hero At cave ⇒
21
```

Figure 1. Example of correct ligatures support in Sublime Text.

Callouts Support

Now it's time to test how callouts interact with the syntax highlighter.

```
The shore IsA location ❶
  Name Shore of Great Sea ❷
  Description "A beautifyl sea shore, probably Paradise lost."
End The shore.
```

- ❶ Instance declaration.
- ❷ Instance name.

BNF Syntax

Currently, BNF blocks are not being syntax highlighted but implementing a syntax definition would be very easy.

```
attribute_declaration = id
                        | 'NOT' id
                        | id integer
                        | id string
                        | id id
                        | id '{' values '}'
```

```
forms = indefinite | definite | negative

definite = 'DEFINITE' article_or_form

indefinite = [ 'INDEFINITE' ] article_or_form

negative = 'NEGATIVE' article_or_form

article_or_form = 'ARTICLE' {statement}
                 | 'FORM' {statement}
```


Part III. Customizing The Look of Code

Sometimes we need to customize how the code is presented to the reader, beyond what syntax highlighting automatically provides.

This part is dedicated to show how this can be achieved and how it looks like in the output of the different backends.

Custom Color-Marking

The *ALAN Beginner's Guide* uses this technique to mark specific lines of code, or to single out certain parts of a line, by adopting a custom coloring notation to show which code was added in each step of a tutorial, or just to draw the reader's attention to specific parts of the code.

Yellow Highlighting

To mark part of the code as if it was highlighted with a yellow marker, we can enclose it within a pair of single or double hash symbols (#).

```
-- Custom coloring.  
-- Example 1: Yellow highlighting.  
Every book IsA object.  
  Description "It's just a book."  
  Has not been_read.  
End Every.
```



Currently the PDF toolchain doesn't support correct styling of the `marked` element, it will be only rendered in italics. I'm still trying to work out how to style this in the custom FOPUB template.

Green Highlighting

To use a green color, the `[green]#` variant must be used instead, which uses a custom defined style.

```
-- Custom coloring.  
-- Example 2: green highlighting.  
Every book IsA object.  
  Description "It's just a book."  
  Has not been_read.  
End Every.
```



Currently the PDF toolchain doesn't support styling this custom inline element at all, therefore it won't produce any visible result in the PDF document. I'm still trying to work out how to support styling this element.

