编写语法突出显示文件

2005年3月24日 |克里斯托夫·卡尔曼

注意：有关如何编写语法突出显示文件的最新版本，请参阅 Kate 手册。

提示：如果要编写语法突出显示文件，XML 完成插件可能会很有帮助。

本节概述了 KDE4 中的突出显示定义 XML 格式。基于一个小例子，它将描述主要组件及其含义和用法。下一节将详细介绍突出显示检测规则。

凯特突出显示定义文件的主要部分

正式定义，又名DTD存储在文件中，该文件应安装在系统上的文件夹中。如果未设置，请使用 查找文件夹。 `language.dtd` `$KDEDIR/share/apps/katepart/syntax` `$KDEDIR` `kde4-config --prefix`

示例

突出显示文件包含一个标头，用于设置 XML 版本和文档类型：

```
<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE language SYSTEM "language.dtd">
```

定义文件的根是元素**语言**。可用属性包括：

必需属性：

- **名称** 设置语言的名称。它出现在菜单和对话框中。
- **部分**指定类别。
- **扩展名**定义文件扩展名，如“*.cpp;*。h"

可选属性：

- **哑剧类型** 关联文件 基于哑剧类型。
- **版本** 指定定义文件的当前版本。
- **凯特版本**指定最新支持的凯特版本。
- **区分大小写**的定义，无论关键字是否区分大小写。
- 如果另一个突出显示定义文件使用相同的扩展名，则**优先级**是必需的。优先级越高。
- **作者**包含作者的姓名及其电子邮件地址。
- **许可证**包含许可证，通常是 LGPL、艺术、GPL 等。指定许可证很重要，因为 kate 团队需要一些法律支持来分发文件。
- **隐藏**定义，该名称是否应出现在凯特的菜单中。

所以下一行可能看起来像这样：

```
<language name="C++" version="1.00" kateversion="2.4" section="Sources" extensions="*.cpp;*.h" >
```

接下来是**突出显示**元素，其中包含可选元素**列表**以及必需元素**上下文**和 **itemData。**

列表元素包含关键字列表。在本例中，关键字为*类*和*常量*。您可以根据需要添加任意数量的列表。**上下文**元素包含所有上下文。默认情况下，第一个上下文是突出显示的开始。上下文 *Normal Text* 中有两个规则，它们将关键字列表与名称 *somename* 匹配，另一个规则用于检测引号并将上下文切换为*字符串*。要了解有关规则的更多信息，请阅读下一章。
第三部分是**项目数据**元素。它包含上下文和规则所需的所有颜色和字体样式。
在此示例中，使用 **itemData** *普通文本*、*字符串*和*关键字*。

```
<highlighting> <list name="somename"> <item> class </item> <item> const </item> </list> <contexts> <cc
```

突出显示定义的最后一部分是可选的**常规**部分。它可能包含有关关键字，代码折叠，注释和缩进的信息。

注释部分定义使用什么字符串引入单行注释。您还可以使用具有附加属性*末尾*的*多行定义多行*注释。如果用户按相应的快捷方式进行*注释/取消注释，则使用此选项。*
关键字部分定义关键字列表是否区分大小写。其他属性将在后面解释。

```
<general> <comments> <comment name="singleLine" start="#"/> </comments> <keywords casesensitive="1"/>
```

各节详情

本部分将描述上下文，itemData，
关键字，注释，代码折叠和缩进的所有可用属性。

元素**上下文**属于组**上下文**。上下文本身定义了特定于上下文的规则，例如如果突出显示系统到达一行的末尾，会发生什么。可用属性包括：

- **命名**上下文名称/标识符。规则将使用此名称来指定在规则匹配时要切换到的上下文。
- **属性**，如果当前上下文中没有匹配的规则，则使用该属性。
- **行结束上下文**定义突出显示系统在到达行尾时切换到的上下文。这可能是另一个上下文的名称，**#stay**不切换上下文（即不执行任何操作）或**#pop**这将导致离开此上下文。例如，可以使用 **#pop#pop#pop** 来弹出三次。
- **行开始上下文**定义上下文在遇到行的开头时的上下文。默认值：#stay。
- **下降**定义突出显示系统是否切换到下降上下文中指定的上下文（如果没有匹配的规则）。默认值：*假*。
- 如果没有规则匹配，**则下降上下文**指定下一个上下文。
- **dynamic** 如果为 *true*，则上下文会记住由动态规则保存的字符串/占位符。例如，*对于HERE*文档，这是必需的。默认值：*假*。

元素**项数据**位于组**项数据中**。它定义了字体样式和颜色。因此，可以定义自己的样式和颜色，但是如果可能的话，我们建议坚持使用默认样式，以便用户始终看到不同语言中使用的相同颜色。但是，有时没有其他方法，有必要更改颜色和字体属性。属性名称和 def 样式数字是必需的，其他是可选的。可用属性包括：

- **名称** 设置项目数据的名称。上下文和规则将在其属性中使用此名称。
- **属性**以引用项目数据。
- **定义**要使用的默认样式。可用的默认样式将在后面详细介绍。
- **颜色**定义颜色。有效格式为“#rrggbb”或“#rgb”。
- **sel 颜色**定义选区颜色。
- **如果**为 *true*，则为斜体，则文本将为*斜体*。
- **bold** if *true*, the text will be **bold**.
- **underline** if *true*, the text will be underlined.
- **strikeout** if *true*, the text will be ~~stroked out~~.
- **spellChecking** if true, the text will be spell checked, otherwise it will be ignored during spell check.

The element **keywords** in the group **general** defines keyword properties. Available attributes are:

- **casesensitive** may be *true* or *false*. If *true*, all keywords are matched case sensitive. Default: *true*.
- **weakDeliminator** is a list of characters that do *not* act as word delimiters. For example the dot (**.**) is a word delimiter. Assume a keyword in a **list** contains a dot, it will only match if you specify the dot as a weak delimiter.
- **additionalDeliminator** defines additional delimiters.
- **wordWrapDeliminator** defines characters after which a line wrap may occur. Default delimiters and word wrap delimiters are the characters **.():!+,-<=>%&_/;?[]^{|}~_*, space (' ')** and **tabulator (\t)**.

The element **comment** in the group **comments** defines comment properties which are used for and .
Available attributes are: `Tools > Comment` `Tools > Uncomment`

- **name** is either *singleLine* or *multiLine*.
- If you choose *singleLine* the optinal attribute **position** is available. Default for this attribute is to insert the single line comment string in column 0. If you want it to appear after the whitespaces you have to set it to *afterwhitespace*, like: . `position="afterwhitespace"`
- If you choose *multiLine* the attributes *end* and *region* are required.
- **start** defines the string used to start a comment. In C++ this is . `/*`
- **end** defines the string used to close a comment. In C++ this is . `*/`
- **region** should be the name of the the foldable multiline comment. Assume you have *beginRegion="Comment"* … *endRegion="Comment"* in your rules, you should use *region="Comment"*. This way uncomment works even if you do not select all the text of the multiline comment. The cursor only must be in the multiline comment.

The element **folding** in the group **general** defines code folding properties. Available attributes are:

- **indentationsensitive** if *true*, the code folding markers will be added indentation based, like in the scripting language Python. Usually you do not need to set it, as it defaults to *false*.

The element **indentation** in the group **general** defines which indenter will be used, however we strongly recommend to omit this element, as the indenter usually will be set by either defining a File Type or by adding a mode line to the text file. If you specify an indenter though, you will force a specific indentation on the user, which he might not like at all.
Available attributes are:

- **mode** is the name of the indenter. Available indenters right now are: *none*, *normal*, *cstyle*, *haskell*, *lilypond*, *lisp*, *python*, *ruby* and *xml*.

Available Default Styles

Default styles are predefined font and color styles. For convenience Kate provides several default styles, in detail:

- **dsNormal**, used for normal text.
- **dsKeyword**, used for keywords.
- **dsDataType**, used for data types.
- **dsDecVal**, used for decimal values.
- **dsBaseN**, used for values with a base other than 10.
- **dsFloat**, used for float values.
- **dsChar**, used for a character.

- **dsString**, used for strings.
- **dsComment**, used for comments.
- **dsOthers**, used for ‘other’ things.
- **dsAlert**, used for warning messages.
- **dsFunction**, used for function calls.
- **dsRegionMarker**, used for region markers.
- **dsError**, used for error highlighting and wrong syntax.

Highlight Detection Rules

This section describes the syntax detection rules.

Each rule can match zero or more characters at the beginning of the string they are tested against. If the rule matches, the matching characters are assigned the style or *attribute* defined by the rule, and a rule may ask that the current context is switched.

A rule looks like this:

The *attribute* identifies the style to use for matched characters by name, and the *context* identifies the context to use from here.

The *context* can be identified by:

- An *identifier*, which is the name/identifier of another context.
- An *order* telling the engine to stay in the current context (**#stay**), or to pop back to a previous context used in the string (**#pop**). To go back more steps, the #pop keyword can be repeated: **#pop#pop#pop**

Some rules can have *child rules* which are then evaluated only if the parent rule matched. The entire matched string will be given the attribute defined by the parent rule. A rule with child rules looks like this:

```
<RuleName (attributes)> <ChildRuleName (attributes) /> ... </RuleName>
```

Rule specific attributes vary and are described in the following sections.

Common attributes

All rules have the following attributes in common and are available whenever a **(common attributes)** appears. All following attributes are optional.

- **attribute** maps to a defined *itemData*. Default: the attribute from the *destination context*
- **context** specifies the context to which the highlighting system switches if the rule matches. Default: *#stay*
- **beginRegion** starts a code folding block. Default: unset.
- **endRegion** closes a code folding block. Default: unset.
- **lookAhead**, if *true*, the highlighting system will not process the matches length. Default: *false*.
- **firstNonSpace**, if *true*, the rule only matches, if the string is the first non-whitespace in the line. Default: *false*.
- **column** defines the column. The rule only matches, if the current column matches the given one. Default: unset.

Dynamic rules

Some rules allow the optional attribute **dynamic** of type boolean that defaults to *false*. If dynamic is *true*, a rule can use placeholders representing the text matched by a *regular expression* rule that switched to the current context in its **string** or **char** attributes. In a **string**, the placeholder **%N** (where N is a number) will be replaced with the corresponding capture **N** from the calling regular expression. In a **char** the placeholer must be a number **N** and it will be replaced with the first character of the corresponding capture **N** from the calling regular expression. Whenever a rule allows this attribute it will contain a *(dynamic)*.

- **dynamic** may be either *true* or *false*. Default: *false*.

The Rules in Detail

DetectChar

Detect a single specific character. Commonly used for example to find the ends of quoted strings.

```
<DetectChar char="(character)" (common attributes) (dynamic) />
```

The **char** attribute defines the character to match.

Detect2Chars

Detect two specific characters in a defined order.

```
<Detect2Chars char="(character)" char1="(character)" (common attributes) (dynamic) />
```

The **char** attribute defines the first character to match, **char1** the second.

AnyChar

Detect one character of a set of specified characters.

```
<AnyChar String="(string)" (common attributes) />
```

The **String** attribute defines the set of characters.

StringDetect

Detect an exact string.

```
<StringDetect String="(string)" [insensitive="true|false"] (common attributes) (dynamic) />
```

The **String** attribute defines the string to match. The **insensitive** attribute defaults to *false* and is passed to the string comparison function. If the value is *true* insensitive comparing is used.

WordDetect (KDE >= 4.5, Kate >= 3.5)

Detect an exact string but additionally require word boundaries like a dot (.) or a whitespace on the beginning and the end of the word. You can think of $|b|b$ in terms of a regular expression.

```
<WordDetect String="(string)" [insensitive="true|false"] (common attributes) (dynamic) />
```

The **String** attribute defines the string to match. The **insensitive** attribute defaults to *false* and is passed to the string comparison function. If the value is *true* insensitive comparing is used.

RegExpr

Matches against a regular expression.

```
<RegExpr String="(string)" [insensitive="true|false"] [minimal="true|false"] (common attributes) (dynam
```

- The **String** attribute defines the regular expression.
- **insensitive** defaults to *false* and is passed to the regular expression engine.
- **minimal** defaults to *false* and is passed to the regular expression engine.

Because the rules are always matched against the beginning of the current string, a regular expression starting with a caret (**^**) indicates that the rule should only be matched against the start of a line.

keyword

Detect a keyword from a specified list.

```
<keyword String="(list name)" (common attributes) />
```

The **String** attribute identifies the keyword list by name. A list with that name must exist.

Int

Detect an integer number.

```
<Int (common attributes) (dynamic) />
```

This rule has no specific attributes. Child rules are typically used to detect combinations of **L** and **U** after the number, indicating the integer type in program code. Actually all rules are allowed as child rules, though, the DTD only allowes the child rule **StringDetect**.
The following example matches integer numbers follows by the character ‘L’.

```
<Int attribute="Decimal" context="#stay" > <StringDetect attribute="Decimal" context="#stay" String="L"
```

Float

Detect a floating point number.

```
<Float (common attributes) />
```

This rule has no specific attributes. **AnyChar** is allowed as a child rules and typically used to detect combinations, see rule **Int** for reference.

HlCOct

Detect an octal point number representation.

```
<HlCOct (common attributes) />
```

This rule has no specific attributes.

HlCHex

Detect a hexadecimal number representation.

```
<HlCHex (common attributes) />
```

This rule has no specific attributes.

HlCStringChar

Detect an escaped character.

```
<HlCStringChar (common attributes) />
```

This rule has no specific attributes.

It matches literal representations of characters commonly used in program code, for example **\n** (newline) or **\t** (tabulator).

The following characters will match if they follow a backslash (***): **abefnrtv"'?*. Additionally, escaped hexadecimal numbers like for example **\xff** and escaped octal numbers, for example **\033** will match.

HlCChar

Detect an C character.

```
<HlCChar (common attributes) />
```

This rule has no specific attributes.

It matches C characters enclosed in a tick (Example: '**c**'). So in the ticks may be a simple character or an escaped character. See HlCStringChar for matched escaped character sequences.

RangeDetect

Detect a string with defined start and end characters.

```
<RangeDetect char="(character)" char1="(character)" (common attributes) />
```

char defines the character starting the range, **char1** the character ending the range. Usefull to detect for example small quoted strings and the like, but note that since the highlighting engine works on one line at a time, this will not find strings spanning over a line break.

LineContinue

Matches a backslash ('&#8217;) at the end of a line.

```
<LineContinue (common attributes) />
```

This rule has no specific attributes.
This rule is useful for switching context at end of line, if the last character is a backslash (‘****’). This is needed for example in C/C++ to continue macros or strings.

IncludeRules

Include rules from another context or language/file.

```
<IncludeRules context="contextlink" [includeAttrib="true|false"] />
```

The **context** attribute defines which context to include.
If it a simple string it includes all defined rules into the current context, example:

```
<IncludeRules context="anotherContext" />
```

If the string begins with **##** the highlight system will look for another language definition with the given name, example:

```
<IncludeRules context="##C++" />
```

If **includeAttrib** attribute is *true*, change the destination attribute to the one of the source. This is required to make for example commenting work, if text matched by the included context is a different highlight than the host context.

DetectSpaces

Detect whitespaces.

```
<DetectSpaces (common attributes) />
```

This rule has no specific attributes.
Use this rule if you know that there can several whitespaces ahead, for example in the beginning of indented lines. This rule will skip all whitespace at once, instead of testing multiple rules and skipping one at the time due to no match.

DetectIdentifier

Detect identifier strings (as a regular expression: [a-zA-Z_][a-zA-Z0-9_]*).

```
<DetectIdentifier (common attributes) />
```

This rule has no specific attributes.
Use this rule to skip a string of word characters at once, rather than testing with multiple rules and skipping one at the time due to no match.

Tips & Tricks

Once you have understood how the context switching works it will be easy to write highlight definitions. Though you should carefully check what rule you choose in what situation. Regular expressions are very mighty, but they are slow compared to the other rules. So you may consider the following tips.

- If you only match two characters use **Detect2Chars** instead of **StringDetect**. The same applies to **DetectChar**.
- Regular expressions are easy to use but often there is another much faster way to achieve the same result. Consider you only want to match the character **#**if it is the first character in the line. A regular expression based solution would look like this:

  You can achieve the same much faster in using:

  ```
  <DetectChar attribute="Macro" context="macro" char="#" firstNonSpace="true" />
  ```

  If you want to match the regular expression '**^#**' you can still use **DetectChar** with the attribute **column="0"**. The attribute **column** counts character based, so a tabulator still is only one character.

  - You can switch contexts without processing characters. Assume that you want to switch context when you meet the string **/*, but need to process that string in the next context. The below rule will match, and the lookAhead**attribute will cause the highlighter to keep the matched string for the next context.
  - Use **DetectSpaces** if you know that many whitespaces occur.
  - Use **DetectIdentifier** instead of the regular expression '**[a-zA-Z_]\w***'.
  - Use default styles whenever you can. This way the user will find a familiar environment.
  - Look into other XML-files to see how other people implement tricky rules.
  - You can validate every XML file by using the command `xmllint --dtdvalid language.dtd mySyntax.xml` .
  - If you repeat complex regular expression very often you can use *ENTITIES*. Example:]>

    Now you can use *&myref;* instead of the regular expression.

Products

Plasma

KDE Applications

KDE Frameworks

Plasma Mobile

KDE Neon

Develop

Techbase Wiki

API Documentation

Qt Documentation

Inqlude Documentation

KDE Goals

Source code

News & Press

Announcements

KDE.news

Planet KDE

Press Contacts

Miscellaneous Stuff

Thanks

Resources

Community Wiki

UserBase Wiki

Support

Download KDE Software

Code of Conduct

Privacy Policy

Applications Privacy Policy

Destinations

KDE Store

KDE e.V.

KDE Free Qt Foundation

KDE Timeline

KDE Manifesto

International Websites