19.7.1 Klasse TextHighlighter (gb.highlight)

Diese Klasse ist die Elternklasse aller Text-Highlighter in gb.highlight.

Sie kann wie ein Objekt verwendet werden, indem bei Bedarf eine versteckte Instanz erzeugt wird.

Die Klasse ermöglicht die Erzeugung eines benutzerdefinierten Text-Highlighters auf der Basis einer Definitionsdatei *.highlight.

19.7.1.1 Eigenschaften

Die Klasse TextHighlighter verfügt über die nachstehenden Eigenschaften:

Eigenschaft	Datentyp	Beschreibung
CanRewrite	Boolean	Gibt zurück oder setzt, ob der Text-Highlighter den zu hervorzuhebenden Text neu schreiben darf.
Comment	Boolean	Gibt zurück oder setzt, ob der hervorgehobene Text ein Kommentar ist.
Key	String	Gibt den internen Namen des Text-Highlighters zurück.
Name	String	Gibt den Anzeigenamen des Text-Highlighters zurück.
Keywords	String[ ]	Gibt die Liste der Schlüsselwörter zurück, die mit dem Text-Highlighter verknüpft sind.
LengthAdded	Integer	Gibt die Anzahl der hinzugefügten (oder entfernten) Zeichen zurück, nachdem der Text umgeschrieben wurde.
Limit	Boolean	Gibt zurück oder setzt, ob der hervorgehobene Text eine Funktionsgrenze ist.
States	String[ ]	Gibt die Liste der vom Text-Highlighter verwendeten Bezeichner für die hervorzuhebenden Textelemente (Token) zurück.
TextAfter	String	Den hervorgehobenen Text zurückgeben oder setzen, nachdem er umgeschrieben worden ist.

Tabelle 19.7.1.1.1 : Eigenschaften der Klasse TextHighlighter

19.7.1.2 Methoden

Die Klasse TextHighlighter besitzt diese Methoden:

Methode	Rückgabetyp	Beschreibung
Paint (Text As String, X As Float, Y As Float [, Theme As TextHighlighterTheme, Pos As Integer])	-	Zeichnet den hervorgehobenen Text. Text: Der hervorzuhebende Text. X, Y: Die Koordinaten des Startpunkts der Zeichnung. Theme: Das zu verwendende Thema. Wenn nicht angegeben, so wird ein Standardthema verwendet. Pos: Die Position des ersten zu zeichnenden Zeichens. Standardmäßig wird der gesamte Text gezeichnet.
RegisterState (Name As String)	Integer	Registriert einen Text-Highlighter. Name: Die Funktion gibt einen Index zurück, der zum Füllen des Highlight-Arrays verwendet werden muss.
Run (Text As String, State As Short[ ])	Byte[ ]	Markiert einen Textabschnitt. Text: Der Text, der hervorgehoben werden soll. In der Regel muss es sich um eine vollständige Textzeile handeln, die mit einem Zeilenumbruch endet. Status: Der interne Zustand des Text-Highlighters. Gibt das Ergebnis der Hervorhebung als ein Array von Bytes zurück. Der interne Zustand muss mit einem leeren Array von Bytes initialisiert werden. Wenn die Run()-Methode zurückkehrt, wird aktualisiert, um den Zustand des Text-Highlighters nach der Hervorhebung wiederzugeben, so dass Sie die Run()-Methode erneut aufrufen können.
ToANSI (Text As String [, The-	String	Gibt das Ergebnis der Texthervorhebung als VT100 ANSI-

Methode	Rückgabetyp	Beschreibung
me As TextHighlighterTheme])		Zeichen zurück, die auf einem Terminal ausgedruckt werden können. Text: Der hervorzuhebende Text. Theme: Das zu verwendende Thema. Wenn es nicht angegeben wurde, so wird ein Standardthema verwendet.
ToHTML (Text As String [, Theme As TextHighlighterTheme])	String	Gibt das Ergebnis der Texthervorhebung als HTML zurück. Text: Der hervorzuhebende Text. Theme: Das zu verwendende Thema. Wenn es nicht angegeben wurde, so wird ein Standardthema verwendet.
ToRichText (Text As String [, Theme As TextHighlighterTheme])	String	Gibt das Ergebnis der Texthervorhebung als Rich Text zurück. Text: Der hervorzuhebende Text. Thema: Das zu verwendende Thema. Wenn nicht angegeben, wird ein Standardthema verwendet.

Tabelle 19.7.1.2.1 : Methoden der Klasse TextHighlighter

19.7.1.3 Entwicklung eines neuen Text-Highlighters

Am Markgraf-Albrecht-Gymnasium in Osterburg wurde für den Informatik-Unterricht in der Abiturstufe die Programmiersprache LIPA (Language for an internally program-controlled automat) entwickelt. Für den Quelltext-Editor sollte ein eigener Text-Highlighter entwickelt werden, um die Lesbarkeit der Quelltexte zu erhöhen.

Der einfache Befehlssatz der Sprache ist überschaubar. Daher sind in der folgenden Tabelle die frei gewählten Farben für die Befehlsgruppen, Kommentare und Adressen abgebildet:

Gruppe	Befehl	Farbe
Eingabe-Ausgabe-Befehle	INP(xx), OUT(xx)	
Transport-Befehle	LDA(xx), STA(xx)	
Arithmetische Befehle	INC(), DEC()	
Sprung-Befehle	JPU(xx), JPP(xx), JPZ(xx), JPN(xx)	
Ende-Befehl	EOJ()	
Kommentare	{ … }	
Ganzzahlige Adressen (0...99)	xx	

Tabelle 19.7.1.3.1 : Einfacher Befehlssatz

Alle Befehle – mit Ausnahme von EOJ() – enden mit einem Semikolon; optional gefolgt von einem Kommentar in geschweiften Klammern.

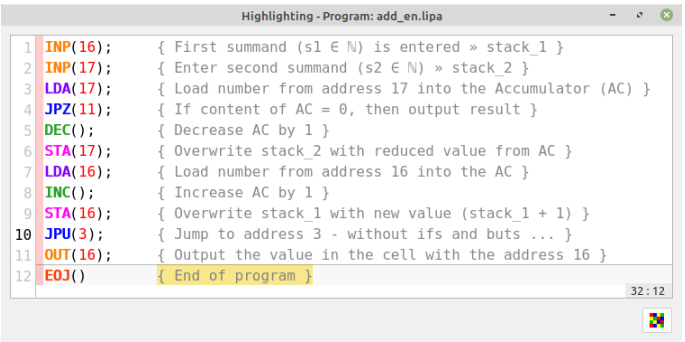

Abbildung 19.7.1.3.1: Quelltext-Editor für LIPA mit Text-Highlighting

Auf die Sprache LIPA und deren Text-Highlighter wird in den folgenden Kapiteln mehrfach Bezug genommen.

19.7.1.4 Definitionsdatei für den Text-Highlighter LIPA

Eine gute Vorlage für eigene Definitionsdateien *.highlight bieten die Definitionsdateien im Gambas Quelltext unter .../gambas-3.19.5/comp/src/gb.highlight/highlight.

Es folgt der Inhalt einer ersten Version der Definitionsdatei lipa.highlight eines Text-Highlighters für die Programmiersprache LIPA, der den Vorgaben aus der obigen Tabelle folgt und den Sie mit einem Editor Ihrer Wahl wie xed unter Beachtung der Hinweise in der Dokumentation schreiben und nach der Erprobung in einem geeigneten Pfad abspeichern können!

```
Number{NumberStyle=Number}:
  match /[0-9]+/
Jump{JumpStyle}:
  keyword JPN JPZ JPP JPU
Comment:
  from { to }
#------------------------
InOut:
  keyword INP OUT
LoadA:
  keyword LDA
Store:
  keyword STA
Operation:
  symbol - +
  keyword DEC INC
End:
  keyword EOJ
```

19.7.1.5 Hinweise zum Inhalt einer Definitionsdatei

In einer Definitionsdatei *.lighlight müssen Sie in so genannten Hervorhebungsregeln,

- zuerst festlegen, welchen Bezeichner die einzelnen Hervorhebungsregeln erhalten,
- dann optional in geschweiften Klammern einen Style-Namen vergeben,
- danach optional nach einem Gleichheitszeichen den Bezeichner für einen Style angeben. Dieser kann ein Bezeichner aus der u.a. Liste von fest vorgegebenen Style-Bezeichnern sein, die in allen Hervorhebungsthemen definiert sind. Diese können Sie problemlos in jeder benutzerdefinierten Definitionsdatei *.highlight verwenden. Oder die Style-Bezeichner werden von Ihnen frei vergeben wie zum Beispiel 'InputStyle' und
- abschließend vorgeben, für welchen Textabschnitt im Text-Editor ein bestimmter Style gelten soll.

Auszug aus der Liste der in Gambas vorgegebenen Style-Bezeichner (Standard-Styles)

```
Normal, Added, Removed, Error, Comment, Documentation, Keyword, Function, Operator, Symbol, Number, String,
Datatype, Preprocessor, Escape und Constant.
```

Jede Zeile in einer Definitionsdatei *.highlight, die mit einem Doppelpunkt **:** endet, leitet eine neue Hervorhebungsregel ein. Die Syntax lautet:

```
'Bezeichner der Hervorhebungsregel' [ { Style-Name [ = 'Bezeichner des Styles' ] } ] :
```

Beispiel 1

```
Number{NumberStyle=Number}:
```

definiert die Hervorhebungsregel mit dem Bezeichner 'Number', die mit dem frei gewählten Style-Namen 'NumberStyle' verknüpft ist. Da 'NumberStyle' ein neuer Style-Name ist, wird der Highlighter im angegebenen Beispiel angewiesen, den (bereits existierenden) Style 'Number' mit dem Farbwert #E62222 aus dem Gambas-Thema zu verwenden.

Beispiel 2

```
Jump{JumpStyle}:
```

definiert die Hervorhebungsregel mit dem Bezeichner 'Jump', die mit dem frei gewählten Style-Namen 'JumpStyle' verknüpft ist. Da 'JumpStyle' ein neuer Style-Name ist, aber kein Style festgelegt wurde,

wird der Highlighter angewiesen, automatisch den Style 'Normal' mit dem Farbwert #000000 zu verwenden. Sie können später einen benutzer-definierten Style definieren und ihn unter dem Style-Namen 'JumpStyle' in das aktuelle Highlighter-Thema einfügen. Damit wird der bisher verwendete Style 'Normal' überschrieben.

Beispiel 3

```
Comment:
```

definiert die Hervorhebungsregel mit dem Bezeichner 'Comment', die automatisch mit dem gleichnamigen (Standard)-Style-Namen 'Comment' verknüpft wird, zu dem auch ein Standard-Style mit dem Farbwert #888786 vorhanden ist.

Beispiel 4

```
InOut:
```

definiert die Hervorhebungsregel mit dem Bezeichner 'InOut', die automatisch mit dem Style-Namen 'InOut' verknüpft wird. Da 'InOut' ein neuer Style-Name ist, aber dazu kein Standard-Style existiert, wird der Highlighter angewiesen, automatisch den Style 'Normal' mit dem Farbwert #000000 zu verwenden. Sie können später einen benutzer-definierten Style definieren und ihn unter dem Style-Namen 'InOut' in das aktuelle Highlighter-Thema einfügen. Damit wird der bisher verwendete Style 'Normal' überschrieben.

Gut zu wissen: Einzeilige Kommentare in einer Definitionsdatei beginnen mit einem #-Zeichen.

Nach dem ersten Teil der Definition einer Hervorhebungsregel folgen stets ein oder mehrere Befehle. Sie legen fest, für welchen Textabschnitt oder für welche Textabschnitte im Text der definierte Style gelten soll. Diese Befehle stehen jeweils in einer Zeile und müssen mit mindestens einem Leerzeichen eingerückt werden.

Beispiele

```
Number{NumberStyle=Number}:
  match /[0-9]+/
Jump{Jump}:
  keyword JPN JPZ JPP
Operation:
  symbol - +
  keyword DEC INC
Comment:
  from { to }
```

Nachfolgend eine Tabelle der Syntax möglicher Befehle, die Sie verwenden können. Bei Pattern und Muster kann es sich um einen einfachen Ausdruck oder einen regulären Ausdruck handeln.

Befehl	Beschreibung
match pattern	Gesucht wird nach dem Textabschnitt, der auf das angegebene Muster passt. Beachten Sie, dass 'pattern' in Schrägstrichen // eingefasst wird.
word word#1 word#2 ... word#n	Gesucht wird nach den in der Liste angegebenen Wörtern.
keyword word#1 word#2 ... word#n	Gesucht wird nach den in der Liste angegebenen Schlüsselwörtern. Schlüsselwörter werden (intern) zu einer Liste hinzugefügt, die ausgelesen werden können.
symbol symbol#1 symbol#2 ... symbol#n	Gesucht wird nach den in der Liste angegebenen Symbolen, die jeweils aus einem Zeichen bestehen.
from start_pattern to end_pattern	Gesucht wird nach dem Textabschnitt, der zwischen Startmuster und Endmuster liegt.
from start_pattern	Gesucht wird nach dem Textabschnitt, auf den das angegebene Startmuster passt.
from here to end_pattern	Gesucht wird nach dem Textabschnitt, auf den das angegebene Muster passt.

Befehl	Beschreibung
from here	Gesucht wird nach dem Textabschnitt, auf den das angegebene Muster passt.
between start_pattern and end_pattern	Gesucht wird nach dem Textabschnitt, auf den das angegebene Muster passt.
between start_pattern	Gesucht wird nach dem Textabschnitt, auf den das angegebene Muster passt.
between here and end_pattern	Gesucht wird nach dem Textabschnitt, der von der Startposition an auf das angegebene Muster passt.

Tabelle 19.7.1.5.1 : Befehle in Hervorhebungsregeln

Wenn Sie in einem LIPA-Programm im folgenden Quelltext-Abschnitt

```
JPZ(11);     { If content of AC = 0, then output result }
JPZ(11);     { If content of AC = 0, then output result }
```

den Kommentar in grauer Farbe hervorheben wollen, dann könnten Sie im ersten Fall den Kommentar-Text inklusive der umfassenden geschweiften Klammern grau einfärben oder im zweiten Fall nur den Kommentar-Text innerhalb der geschweiften Klammern einfärben. Im ersten Fall wäre die komplette Hervorhebungsregel

```
Comment:
  from { to }
```

Für den zweiten Fall gelingt das nur mit diesen verschachtelten Hervorhebungsregeln und Befehlen:

```
# Based on a proposal by Benoit Minisini

CommentBrackets{Operator}:
  from { to }
  CommentContents{Comment}:
    between here and }
```

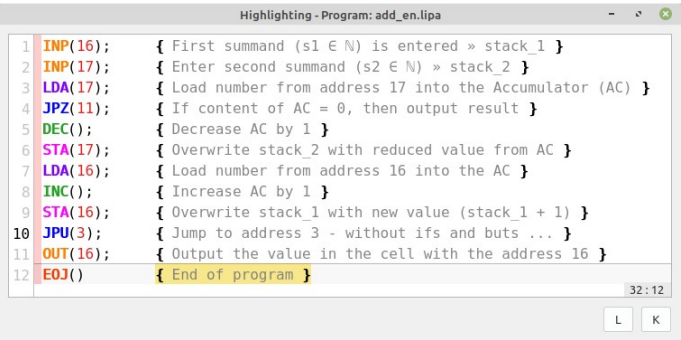

Abbildung 19.7.1.5.1: Quelltext-Editor mit speziellem Text-Highlighting für Kommentare (Fall 2)

Wenn Sie viele verschachtelte Hervorhebungsregeln in Ihrer Definitionsdatei haben, können Sie ausgewählte Abschnitte in einzelne Dateien auslagern, um die Übersichtlichkeit der Definitionsdatei zu verbessern. Als Beispiel wird wieder die Definitionsdatei lipa.highlight präsentiert. Beachten Sie jedoch die geänderte Diktion im Vergleich zur Definitionsdatei am Anfang des Kapitels!

```
@include number.regexp
#--------------------
Comment:
  from { to }
InOut{InOut=Normal}:
  keyword INP OUT
Jump{Jump}:
  keyword JPN JPZ JPP JPU
LoadA:
  keyword LDA
Store:
  keyword STA
Operation:
  symbol - +
```

```
  keyword DEC INC ADD SUB MUL DIV
End:
  keyword EOJ
Number:
# Make sure to enclose the variable $(NUMBER) in slashes
  match /$(NUMBER)/
```

Das ist der Inhalt der Text-Datei number.regex, bei dem die Variable $(NUMBER) über eine @Include-Anweisung in die Definitionsdatei lipa.highlight eingebunden wird:

```
$(NUMBER)=[0-9]+
```

19.7.1.6 Erzeugung eines neuen Text-Highlighters

Bevor Sie einen neuen Text-Highlighter auf der Basis einer gespeicherten Definitionsdatei *.highlight erzeugen können, müssen Sie den Text-Highlighter mit der statische Methode Register(..) registrieren:

```
Static Sub Register ( Key As String [ , Name As String, Path As String ] )
```

Dabei gilt: Key: Interner Name des Text-Highlighters, Name (optional): Anzeigename der Text-Highlighters und Path (optional): Pfad zur Definitionsdatei.

Registrierung des Text-Highlighters mit der Definitionsdatei lipa.highlight:

```
If Not TextHighlighter.List.Exist("lipa") Then
   TextHighlighter.Register("lipa", "LIPA", "./highlight/lipa.highlight")
Endif
```

Anschließend können Sie einen neuen Text-Highlighter erzeugen:

```
Private hTextHighlighter As TextHighlighter
hTextHighlighter = TextHighlighter [ Key As String ]
```

'Key' ist der interne Name des neuen Text-Highlighters:

```
hTextHighlighter = TextHighlighter["lipa"]
```

Eine Liste mit allen registrierten Text-Highlightern können Sie sich in der Konsole der IDE mit der statischen List()-Methode ausgeben lassen:

```
Print "List of all currently registered text highlighters:"
Print String$(51, "-")
Print TextHighlighter.List.Join()
```

Ausgabe:

```
List of all currently registered text highlighters:
---------------------------------------------------
c,cplusplus,css,diff,gambas,highlight,html,javascript,lipa,sh,sql,webpage
```

Hinweis: Die statische Funktion 'Static Function GetName (Key As String) As String' gibt den Anzeigenamen eines registrierten Text-Highlighters mit seinem internen Namen als Argument zurück:

Beispiel

```
Print TextHighlighter.GetName("lipa")
```

Das ist die Ausgabe in der Konsole der IDE:

```
LIPA
```

19.7.1.7 Ergänzungen und Änderungen

Die bisherige Definitionsdatei lipa.highlight des Text-Highlighters für die Programmiersprache LIPA enthält Passagen, bei denen bei der Festlegung die Bezeichner der Hervorhebungsregeln und der Style-Namen keine einheitliche Diktion verwendet wurde und auf im Hintergrund agierende Automatismen gesetzt wurde.

Mit den Erfahrungen bei der Erzeugung und Erprobung des LIPA-Text-Highlighters wird eine überarbeitete Version der Definitionsdatei lipa.highlight angegeben, die auch in den weiteren Kapiteln genutzt wird:

```
InOut{InOutStyle}:
  keyword INP OUT
LoadA{LoadAStyle}:
  keyword LDA
Store{StoreStyle}:
  keyword STA
Jump{JumpStyle}:
  keyword JPN JPZ JPP JPU
Operation{OperationStyle}:
  symbol - +
  keyword DEC INC
End{EndStyle}:
  keyword EOJ
#-------------------------
Address{AddressStyle}:
  match /[0-9]+/
Comment{CommentStyle}:
  from { to }
```

- Die Bezeichner der Hervorhebungsregel kennzeichnen die Befehlsgruppe nun genauer.
- Es wird konsequent für die Style-Namen der Bezeichner der jeweiligen Hervorhebungsregel mit nachfolgenden `Style` eingesetzt, da jeder Style neu festgelegt wird. Auf Standard-Styles wird verzichtet, da diese nur selten passen und dann meist doch überschrieben werden müssen.