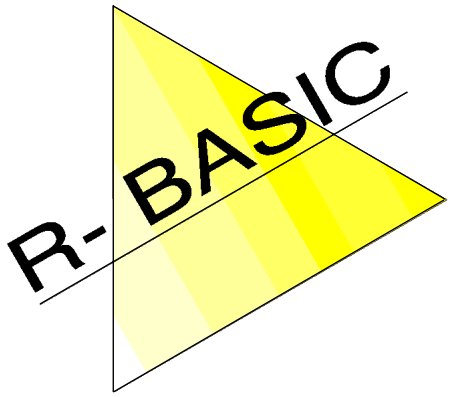
R-BASIC

Einfach unter PC/GEOS programmieren



Neue Funktionen

Version 1.0.3

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

1 Versionsübersicht

Diese Übersicht enthält die wesentlichen Neuerungen, Änderungen und Bugfixes für die R-BASIC-Versionen, die nach der Version 1.0 (Dezember 2021) veröffentlicht wurden und die direkt das Programmieren betreffen. Kleine Anpassungen und Fehlerkorrekturen, die es in jeder Version gibt, oder Verbesserungen des Editor-Handlings, sind nicht explizit aufgeführt.
Falls erforderlich wurde jeweils die Handbücher angepasst.

Version 1.0.4 - Dezember 2024

- Diverse BugFixes: z.B. Crash beim Compilieren sehr großer Programme, Memo und Inputline handlen anfänglichen Font und Textgröße richtig und einige mehr.
- Fortschrittsbox für Compile-Vorgang und diverse weitere Verbesserungen an der UI.
- Anpassungen an neue FM-Sounds für GEOS 6.

Version 1.0.3 - Mai 2023

- BugFix: Funktionsaufruf im RETURN-Statement konnte zu fehlerhaften Ergebnissen oder Systemabsturz führen
- Handling beim Überschreiten des Zahlenbereichs von vorzeichenbehafteten Integer-Zahlen (Integer, LongInt) an das Vorgehen anderer Programmiersprachen angepasst: Keine Begrenzung auf Maximalwert mehr, sondern einfache Übertragsbildung.

Version 1.0.2 - November 2022

- Tab-Styled ItemGroups bei RadioButtons ergänzt
- Unterstützung macOS und Linux: Zeilentrenner LF und erweiterte Konvertierungsfunktionen
- BugFix: Launcher (R-App) registriert sich bei Neustart wieder für das Clipboard.
- Suche in Textobjekten implementiert
- WizzardEditor: Helpfile verständlicher formuliert

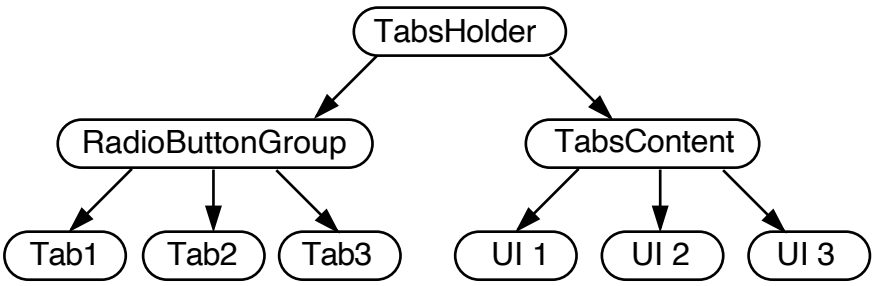
Version 1.0.1 - Februar 2022

- Bugfix bei Setzen der Tausender-Punkte
- Bug in TxtObj.WriteToVMFile und WriteToFile bei leeren Texten beseitigt
- Paragraph-Attribute von Memo/InputLine funktionierten nicht

2 Tab-styled RadioButtonGroups

Tabs sind der übliche Weg, Informationen gruppiert darzustellen. Unter GEOS benötigen wir dafür die folgenden Objekte:

- Eine RadioButtonGroup mit den entsprechenden RadioButton-Objekten. Diese Objekte bilden die eigentlichen Tabs. Dazu wird die Instancevariable look auf einen der unten erwähnten Werte gesetzt.
- Ein Group-Objekt, TabsContent genannt, das die zu den verschiedenen Tabs gehörende UI verwaltet.
- Ein Group-Objekt, TabsHolder genannt, das die Anordnung der RadioButton-Group und des TabContent-Objekts organisiert.
- Die eigentliche UI, mit der der Nutzer innerhalb der Tabs interagieren soll.



Einstellungen für die RadioButtonGroup

Die Einstellungen der RadioButtonGroup beschränken sich darauf, die Instancevariable look mit einem der Werte aus der folgenden Tabelle zu belegen und einen der Hints ExpandWidth oder ExpandHeight zu setzen. Außerdem muss ein Apply-Handler definiert werden, der die zum angewählten Tab gehörende UI sichtbar macht und die nicht dazu gehörende UI verbrigt.

Konstante	Wert	hex.	Position der Tabs
LOOK_TABS_TOP	8	&h8	Über dem TabsContent
LOOK_TABS_LEFT	16	&h10	Links vom TabsContent
LOOK_TABS_RIGHT	32	&h20	Rechts vom TabsContent
LOOK_TABS_BOTTOM	64	&h40	Unter dem TabsContent

Hinweis: Wenn sich die Tabs links oder rechts vom TabsContent befinden, sollten Sie für die RadioButton-Objekte den Hint **fixedSize** setzen, weil sie sonst eventuell nicht bis an das TabsContent-Objekt heranreichen. Der Hint ExpandWidth funktioniert (in der aktuellen GEOS-Version) hier nicht.

Intern werden beim Belegen der Instancevariablen look zusätzliche Instancevariablen belegt, die im Folgenden aufgeführt sind.

Konstante	
LOOK_TABS_TOP	ExpandWidth orientChildren = ORIENT_HORIZONTALLY
LOOK_TABS_LEFT	ExpandHeight orientChildren = ORIENT_VERTICALLY
LOOK_TABS_RIGHT	ExpandHeight orientChildren = ORIENT_VERTICALLY justifyChildren = J_RIGHT
LOOK_TABS_BOTTOM	ExpandWidth orientChildren = ORIENT_HORIZONTALLY justifyChildren = J_BOTTOM

Codebeispiel. Den vollständigen Code finden Sie in der Beispieldatei "Tabs Tester" im Ordner "Beispiel\Objekte\Listen".

```
RadioButtonGroup TabsGroup
  Children = TabText, TabNumber, TabButton
  ApplyHandler = TabsChanged ' as ListAction
  selection = 1
  look = LOOK_TABS_TOP
  End OBJECT
```

```
LISTACTION TabsChanged

  TabsNumber.HideDelayed
  TabsMemo.HideDelayed
  TabsButton.HideDelayed

  ON selection SWITCH
  CASE 1:    TabsMemo.visible = TRUE : End CASE
  CASE 2:    TabsNumber.visible = TRUE : End CASE
  CASE 3:    TabsButton.visible = TRUE : End CASE
  End SWITCH

End ACTION
```

DontCenterTabbedChildren

In älteren GEOS-Versionen, die keine Tabs unterstützen, sieht es eventuell besser aus, wenn die RadioButton-Objekte, die statt der Tabs erscheinen, zentriert sind. Dazu müssen Sie die Instancevariable justifyChildren = J_CENTER setzen. Damit in diesem Fall die Tabs nicht auch zentriert werden, können Sie den Hint DontCenterTabbedChildren verwenden.

Syntax UI- Code: **DontCenterTabbedChildren**

Einstellungen für den TabsHolder

Der TabsHolder muss die RadioButtonGroup und das TabsContent Objekt als Children haben. Die Reihenfolge und die Orientierung der Children hängt davon ab, wo sich die Tabs befinden:

LOOK_TABS_TOP	orientChildren = ORIENT_HORIZONTALLY Children = TabsGroup, TabsContent
LOOK_TABS_LEFT	orientChildren = ORIENT_VERTICALLY Children = TabsGroup, TabsContent
LOOK_TABS_RIGHT	orientChildren = ORIENT_VERTICALLY Children = TabsContent, TabsGroup
LOOK_TABS_BOTTOM	orientChildren = ORIENT_HORIZONTALLY Children = TabsContent, TabsGroup

Zusätzlich sollte der Hint **MinimizeChildSpacing** gesetzt werden, um einen Abstand zwischen TabsGroup und TabsContent zu verhindern.

Häufig ist es sinnvoll, die beiden Hints **ExpandWidth** und **ExpandHeight** zu setzen.

Codebeispiel:

```
Group TabsHolder
  Children = TabsGroup, TabsContent
  orientChildren = ORIENT_HORIZONTALLY
  MinimizeChildSpacing
  ExpandWidth : ExpandHeight
  End OBJECT
```

Einstellungen für das TabsContent-Objekt

Das TabsContent-Objekt sollte seine Größe nicht ändern, wenn die UI darin umgeschaltet wird. Das erreicht man durch setzen beiden Hints **ExpandWidth** und **ExpandHeight** oder durch Belegen der Instancevariablen **fixedSize**.

Sehr oft verwendet das TabsContent-Objekt einen hervorgehobenen Rahmen an drei Seiten, um das typische Aussehen zu erzeugen. Der Rahmen auf der Seite, auf der sich die Tabs befinden, sollte nicht hervorgehoben werden. Das erledigt das RadioButton-Objekt. Um das zu realisieren, unterstützen Group-Objekte die Instancevariable **RaisedFrame**. Ihr wird ein Zahlenwert übergeben, der bestimmt, auf welcher Seite ein hervorgehobener Rahmen gezeichnet wird.

RaisedFrame

RaisedFrame zeichnet einen hervorgehobenen Rahmen an einer oder mehreren Seiten des Objekts.
RaisedFrame ist ein Hint, d.h nicht alle GenericClass Abkömmlinge unterstützen diese Funktion. Er sollte nur für Groups verwendet werden.

Syntax	UI- Code:	**RaisedFrame** = **numWert**
	Lesen:	**<numVar>** = **<obj>** . **RaisedFrame**
	Schreiben:	**<obj>**.**RaisedFrame** = **numWert**

Zur Verwendung mit RaisedFrame sind die folgenden Konstanten definiert. Sie können mit + oder OR verbunden werden.

Konstante	Wert	hex.	Wirkung
RF_TOP	32768	&h8000	Rahmen oben
RF_LEFT	8192	&h2000	Rahmen links
RF_RIGHT	4096	&h1000	Rahmen rechts
RF_BOTTOM	16384	&h4000	Rahmen unten

Hinweis: Wird die Instancevariable RaisedFrame verwendet, so wird der Hint **DrawInBox** ignoriert.

Codebeispiel:

```
Group TabsContent
  Children = TabsMemo, TabsNumber, TabsButton
  RaisedFrame = RF_TOP + RF_RIGHT + RF_BOTTOM
  justifyChildren = J_CENTER
  ExpandWidth : ExpandHeight
  End OBJECT
```

3 Erweiterte Konvertierungsfunktionen für macOS und Linux

Zur Unterstützung von Textdateien, die unter Linux oder Mac OS erstellt wurden, oder dort verwendet werden sollen, wurde die Funktionen **FileWriteLine**, **FileInsertLine**, **FileReplaceLine** und **Convert$** sowie die Text-Objekt Methoden **ReplaceFromFile**, **InsertFromFile** und **WriteToFile** in ihrer Funktion erweitert.
Unter DOS und Windows wird die Kombination CRLF (Codes 13 + 10) zur Zeilenbegrenzung verwendet. Linux und macOS verwenden nur das Zeichen LF (Code 10), während GEOS das Zeichen CR (Code 13) verwendet.

FileReadLine$

Alle FileReadLine-Modi (RLM_CLEAR, RLM_REPLACE_TO_CR, RLM_SET_CR und RLM_DONT_CHANGE) arbeiten bereits problemlos mit Dateien, deren Zeilen-Endezeichen ein CRLF (DOS, Windows) oder ein einfaches LF (Linux, macOS) ist.

FileWriteLine

Zur Unterstützung von Linux- und macOS-Datein wurden drei neue WriteLine-Modi eingeführt:

WLM_APPEND_LF (numerischer Wert: 4): Es wird ein LF-Code angehängt. Eventuell im Text vorhandene Zeilen-Endezeichen werden nicht verändert.

WLM_CR_TO_LF (numerischer Wert: 5): Jeder im Text vorhandene CR-Code wird durch LF ersetzt.

WLM_SET_TO_LF (numerischer Wert: 6): Jeder im Text vorhandene CR-Code wird durch LF ersetzt. Zusätzlich wird ein LF angehängt, falls am Ende noch kein LF steht.

Convert$, ReplaceFromFile, InsertFromFile, WriteToFile

Neues Flag CR_TO_LF (numerischer Wert: 512)
Jedes Auftreten eines "CarriageReturn" (CR, Code 13 bzw. "\r") wird durch ein "LineFeed" (LF, Code 10) ersetzt. Dieses Zeichen wird in Text-Dateien unter Linux und macOS als Zeilenbegrenzung verwendet.

Erweiterte Funktion des Flags CRLF_TO_CR (numerischer Wert: 16)
Neben dem Auftreten der Codefolge CR+LF (Codes 13 und 10) wird auch das Auftreten von LF (Code 10) alleine durch ein einfaches "Carriage-Return" (CR, Code 13 bzw. "\r") ersetzt. Damit können auch Textdateien, die unter Mac OS oder Linux erstellt wurden, eingelesen werden, ohne dass man die Quelle der Datei kennen muss.

Eine vollständige Liste der Konvertiermodi und der verfügbaren Flags finden Sie bei der Beschreibung der Funktion Convert$ im Programmierhandbuch Vol.2, Kapitel 2.4.3.

4 Suchen in Text-Objekten

Das Suchen von Textpassagen in längeren Texten gehört häufig zu den grundlegenden Aufgaben von Programmen. R-BASIC unterstützt die Suche in Strings mit den Funktionen *InStr* (Finden von Textpassagen), *CountStr* (Zählen, wie oft Textpassagen vorkommen) und *ReplaceStr$* (Ersetzen von Textpassagen). Texte in Textobjekten können jedoch länger als die maximale Textlänge in R-BASIC sein (das sind max. 1024 Zeichen). Deswegen unterstützt R-BASIC für alle Textobjekte (Memo, InputLine, LargeText und VisText) die Methoden **FindString** (Finden von Textpassagen), **FindStringBackward** (Rückwärtssuche nach Textpassagen), **CountString** (Zählen, wie oft Textpassagen vorkommen) sowie **ReplaceString** (Ersetzen von Textpassagen).

Die Varianten **FindStringWW**, **FindStringBackwardWW**, **CountStringWW** und **ReplaceStringWW** suchen nach "ganzen Wörtern" (engl. whole words).

Allgemeine Hinweise:

- Für alle Methoden kann der Suchbereich durch die Parameter start und end eingeschränkt werden. Ist er leer (start = end) oder ungültig (start > end, start >= texobj.textLen) so wird nichts gefunden. Es erfolgt hierbei keine Fehlermeldung.
- Es ist zulässig, für den Parameter 'end' einen Wert größer als die aktuelle Textlängen (z.B. 1E9) oder den Wert –1 zu wählen. In diesen Fällen wird der Text bis zum Ende durchsucht.
- Ist der zu suchende String leer "", so wird ebenfalls nichts gefunden. Es erfolgt auch hier keine Fehlermeldung.
- Befindet sich nur ein Teil des zu suchenden Strings im angegebenen Suchbereich, ragt er z.B. über den Suchbereich hinaus, so wird ebenfalls nichts gefunden. Wiederum erfolgt keine Fehlermeldung.
- Die Positions- und Rückgabe-Parameter sind intern vom Typ LongInt. Das bedeutet, dass zu durchsuchende Textgröße bei LargeText-Objekten auf etwa 2 Gigabyte begrenzt ist.

FindString

Die Methode FindString ermittelt die Cursor-Position, ab welcher A$ im Text des Text-Objekts zu finden ist. Der Suchbereich kann eingeschränkt werden.
Standardmäßig wird zwischen Groß- und Kleinschreibung unterschieden (ignoreCase = FALSE).

Syntax: **<numVar>** = **<obj>**.**FindString** **A$** [, **start** [, **end** [, **ignoreCase**]]]
Parameter: A$: String-Ausdruck: der zu findende String
start (optional) Beginn des Suchbereichs. Default: Null
end (optional) Ende des Suchbereichs: Default: –1
Übergeben Sie den Wert –1, wenn der Text bis zum Ende durchsucht werden soll.
ignoreCase (optional) Groß/Kleinschreibung ignorieren (TRUE) oder berücksichtigen (FALSE, Default)
Return: Cursor-Position der Fundstelle (Null entspricht dem Textanfang) bzw. –1, wenn der String A$ nicht gefunden wurde.

Beispiele:

```
DIM pos
  pos = MyText.FindString "Paula"      ' gesamten Text durchsuchen
  pos = MyText.FindString "Paula", 12, 200
        ' Text im Bereich der Zeichenpositionen 12 bis 200
        ' durchsuchen
  pos = MyText.FindString "paulA", 0, –1, TRUE
    ' gesamten Text durchsuchen, Groß/Kleinschreibung ignorieren
```

FindStringBackward

Die Methode FindStringBackward ermittelt die Cursor-Position, ab welcher A$ im Text des Text-Objekts zu finden ist, wobei die Suche am Ende des Suchbereichs beginnt. Wird z.B. nach "Hallo" gesucht, so ist der Rückgabewert die Cursorposition des Buchstabens 'H'. Der Suchbereich kann eingeschränkt werden. Standardmäßig wird zwischen Groß- und Kleinschreibung unterschieden (ignoreCase = FALSE).

Syntax:
<numVar> = **<obj>**.**FindStringBackward** **A$** [, **start** [, **end** [, **ignoreCase**]]]
Parameter: A$: String-Ausdruck: der zu findende String
start (optional) Beginn des Suchbereichs. Default: Null
end (optional) Ende des Suchbereichs: Default: –1
Übergeben Sie den Wert –1, wenn der Text ab dem Ende durchsucht werden soll.
ignoreCase (optional) Groß/Kleinschreibung ignorieren (TRUE) oder berücksichtigen (FALSE, Default)
Return: Cursor-Position der Fundstelle (Null entspricht dem Textanfang) bzw. –1, wenn der String A$ nicht gefunden wurde.

Beispiele:

```
DIM pos
  pos = MyText.FindStringBackward "Paula"
    ' gesamten Text durchsuchen
  pos = MyText.FindStringBackward "paulA", 0, –1, TRUE
    ' gesamten Text durchsuchen, Groß/Kleinschreibung ignorieren
```

CountString

Die Methode CountString ermittelt, wie oft A$ im Text des Text-Objekts zu finden ist. Der Suchbereich kann eingeschränkt werden.
Standardmäßig wird zwischen Groß- und Kleinschreibung unterschieden (ignoreCase = FALSE).

Syntax: **<numVar>** = **<obj>**.**CountString** **A$** [, **start** [, **end** [, **ignoreCase**]]]
Parameter: A$: String-Ausdruck: der zu findende String
start (optional) Beginn des Suchbereichs. Default: Null
end (optional) Ende des Suchbereichs: Default: –1
Übergeben Sie den Wert –1, wenn der Text bis zum Ende durchsucht werden soll.
ignoreCase (optional) Groß/Kleinschreibung ignorieren (TRUE) oder berücksichtigen (FALSE, Default)
Return: Häufigkeit des Auftretens des Strings A$ im Suchbereich.

Hinweis: Bereits gefundene Textstellen werden nicht noch einmal berücksichtigt. Enthält der Text beispielsweise die Zeichenfolge "ahaha", so wird bei der Suche nach "aha" nur ein Auftreten gefunden.

Beispiele:

```
DIM pos
  pos = MyText.CountString, "Paula"        ' gesamten Text
  durchsuchen
  pos = MyText.CountString, "Paula", 10, 30    ' eingeschränkte
  Suche
  pos = MyText.CountString, "paulA", 0, –1, TRUE
    ' gesamten Text durchsuchen, Groß/Kleinschreibung ignorieren
```

ReplaceString

Die Methode ReplaceString realisiert eine "Alles Ersetzen"-Funktion für Text-objekte. Jedes Auftreten der Zeichenfolge A$ wird ohne Nachfrage durch die Zeichenfolge B$ ersetzt. Der Bereich, in dem ersetzt wird, kann eingeschränkt werden.
Standardmäßig wird zwischen Groß- und Kleinschreibung unterschieden (ignoreCase = FALSE).

Syntax: **<obj>**.**ReplaceString** **A$**, **B$** [, **start** [, **end** [, **ignoreCase**]]]
Parameter: A$: String-Ausdruck: der zu findende String
B$: String-Ausdruck: neuer Text
start (optional) Beginn des Suchbereichs. Default: Null
end (optional) Ende des Suchbereichs: Default: –1
Übergeben Sie den Wert –1, wenn der Text bis zum Ende durchsucht werden soll.
ignoreCase (optional) Groß/Kleinschreibung ignorieren (TRUE) oder berücksichtigen (FALSE, Default)

Hinweis: Beachten Sie, dass der Text durch das Ersetzten länger werden kann. Dadurch kann er möglicherweise nicht mehr vom Text-Objekt aufgenommen werden. Darüber erfolgt **keine Fehlermeldung**! Der Suchtext wird dann einfach nicht ersetzt.
Sie können die Methode CountString verwenden, um herauszufinden wie oft der zu ersetzende Text im Suchbereich vorkommt. Außerdem können Sie vor dem Ersetzen die aktuelle Textlänge (Instancevariable textLen) und die maximale Textlänge (Instancevariable maxLen) abfragen.
Für LargeText-Objekte existiert dieses Problem nicht.

Beispiele:

```
DIM count
  count = MyText.CountString "Paula"   ' gesamten Text durchsuchen
  count = MyText.CountString "paulA", 0, –1, TRUE
    ' gesamten Text durchsuchen, Groß/Kleinschreibung ignorieren
```

FindStringWW, FindStringBackwardWW, CountStringWW, ReplaceStringWW

Diese Varianten der Methoden prüfen zusätzlich, ob eine Fundstelle als "ganzen Wort" (engl. whole word) aufgefasst werden kann. Nur dann akzeptieren sie die Fundstelle.
Beispiel: Suche nach "all" in einem Text, der nur aus dem Wort "Hallo" besteht:
Die Methode FindString liefert den Wert 1
Die Methode FindStringWW liefert "nicht gefunden" (–1)

Eine Textstelle zählt als "ganzes Wort", wenn sich davor und danach kein Zeichen befindet, das Teil eines Wortes sein kann. Welche Zeichen Teil eines Wortes sein können, hängt möglicherweise von den Umständen ab. R-BASIC verwendet die folgenden Regeln:

Teil eines Wortes kann sein:

- Die Buchstaben 'A' bis 'Z' und 'a' bis 'z'
- Die Ziffern '0' bis '9'
- Der Unterstrich '_'
- Die Umlaute und Buchstaben mit Akzent aus dem Code-Bereich ab 128 (dezimal) entsprechend der GEOS-Codetabelle. Dazu gehört z.B. die deutschen Umlaute einschließlich dem ß, ausländische Buchstaben, z.B. ñ und Ø, sowie Ligaturen wie Œ oder æ.

Nicht Teil eines Wortes kann z.B. folgendes sein:

- mathematische Symbole, z.B. ≥ oder ∞
- griechische Buchstaben, z.B. Ω oder π
- Währungszeichen

Beachten Sie folgendes:

- Die Zeichen innerhalb eines zu findenden Strings werden nicht geprüft. In diesem Sinne kann ein "ganze Wort" auch Leerzeichen oder Satzzeichen enthalten.
- Der Anfang und das Ende des Texts in einem Textobjekts kann ein "ganzes Wort" begrenzen.
- Für eingeschränkte Suchbereiche gilt das nicht! Die Methoden prüfen die beiden Zeichen links und rechts der Fundstelle, selbst wenn diese sich außerhalb des Suchbereichs befinden.

  Beispiel: Der Text in einem Textobjekt besteht aus dem String "Hallo Welt". Der Suchbereich wird auf den Bereich der Zeichen ’o’ bis ’t’ eingeschränkt. FindStringWW findest in diesem Fall den String "o" nicht, da direkt vor dem ’o’ ein ’l’ steht, dass Teil eines Wortes sein kann. Oder anders gesagt: das ’o’ ist immer noch Teil des Wortes Hallo, also kein eigenständiges Wort.

Weitere Beispiele:
Beachten Sie das führende Leerzeichen im letzten Beispiel!

Text im Textobjekt	Suchtext	Rückgabewert FindString	Rückgabewert FindStringWW
"Fred ist schlau!"	"Fred"	0	0
"Fred ist schlau!"	"Fred ist"	0	0
"Fred ist schlau!"	"ist schlau"	5	5
"Fred ist schlau!"	"red"	1	–1
"Fred ist schlau!"	"schlau!"	9	9
"Fred ist schlau!"	" schlau!"	8	–1

Suchen und Weitersuchen

In vielen Fällen wollen Sie sich nicht mit der ersten Fundstelle zufriedengeben, sondern auch die weiteren Fundstellen finden. Die folgenden Routinen zeigen, wie das prinzipiell gemacht wird. Die Idee ist, dass Sie für die nächste Suche die aktuelle Fundstelle ausklammern. Für eine Vorwärtssuche müssen Sie also den Suchbereich auf den Bereich **nach** der letzten Fundstelle einschränken, für eine Rückwärtssuche müssen Sie den Suchbereich auf den Bereich **vor** der letzten Fundstelle einschränken.
Die FindString~Methoden sind dabei so programmiert, dass Sie selbst keine Sonderfälle berücksichtigen müssen, Sie können also ruhig so lange weitermachen, bis die Meldung "nicht gefunden" kommt.
Ein komplettes Beispiel mit verschiedenen Suchvarianten finden Sie Beispieldatei "Such-Demo" im Ordner "Beispiel\Objekte\Text".

```
SUB FindSimple ()
DIM t$, pos, start

  t$ = SuchText.text$
  start = 0
  pos = FindLargeText.FindString t$ start, -1
  WHILE pos >= 0
    MsgBox "Gefunden auf Position "+Str$(pos)
    start = pos + Len(t$)
    pos = FindLargeText.FindString t$ start, -1
  WEND

  MsgBox "Nicht gefunden"

END SUB 'FindSimple
```

```
SUB FindSimpleBackward ()
DIM t$, pos, sEnd

  t$ = SuchText.text$
  sEnd = -1
  pos = FindLargeText.FindStringBackward t$ 0, sEnd
  WHILE pos >= 0
    MsgBox "Gefunden auf Position "+Str$(pos)
    sEnd = pos
    pos = FindLargeText.FindStringBackward t$ 0, sEnd
  WEND

  MsgBox "Nicht gefunden"

End SUB 'FindSimpleBackward
```

Ein weiterer häufiger Fall ist das Ersetzen des gefundenen Strings mit Nachfrage. Die folgende Function *FindAndReplace* zeigt, wie man das realisieren kann.
Beim Ersetzen von Zeichenketten innerhalb eines Textes kann sich der Text verlängern oder verkürzen. Bei der Vorwärtssuche (aber nicht bei der Rückwärtssuche) muss man das bei der Berechnung der neuen Startposition berücksichtigen. Die Funktion *FindAndReplace* berechnet diese Längenänderung in der Variablen diff. So kann sie gleich die neue Suchposition für den nächsten Aufruf von *FindAndReplace* zurückgeben.
Außerdem zeigt der Code, wie man eine Fundstelle im Text für den Nutzer sichtbar markieren kann.

```
FUNCTION FindAndReplace (textObj AS OBJECT, old$, new$ AS STRING,
  startpos AS REAL) AS REAL
DIM foundPos, newPos, cmd, diff

  foundPos = textObj.FindString old$, startPos, -1
  IF foundPos = -1 THEN RETURN -1          ' Nicht mehr gefunden
```

```
  ' Fundstelle markieren
  textObj.cursorPos = foundPos
  textObj.selectionLen = Len(old$)
  Target = textObj                ' Sichtbarkeit sicherstellen

  ' Nachfragen
  cmd = QuestionBox ("Dieses Auftreten ersetzen?")
  IF cmd = YES THEN
    textObj.ReplaceSelection new$

    ' Neue Startposition berechnen
    ' Die Textlänge kann sich geändert haben
    diff = Len(new$) - Len(old$)
    newPos = foundPos + Len(old$) + diff
  ELSE
    ' nur die neue Startposition berechnen
    newPos = foundPos + Len(old$)
  End IF

  RETURN newPos

End FUNCTION 'FindAndReplace
```

Code-Beispiel: Einfacher Aufruf der Funktion *FindAndReplace*. Jedes Auftreten der Zeichenkette "Paul" wird auf Nachfrage durch "Fred" ersetzt.

```
DIM startpos
  sPos = 0
  REPEAT
    sPos = FindAndReplace (MyTextObj, "Paul", "Fred", sPos)
  UNTIL sPos = -1
```