6.8 Komponente gb.inotify

Mit der Komponente *gb.inotify* von Tobias Boege können Sie in Gambas-Programmen auf die linux-spezifische Schnittstelle "inotify" zugreifen. Informationen zu dieser Schnittstelle finden Sie unter https://wiki.ubuntuusers.de/inotify. Die Schnittstelle ermöglicht es Ihnen, ausgewählte Dateisystem-Ereignisse abzufangen. Sie können so einen Pfad des Dateisystems überwachen, hinter dem eine Datei oder ein Verzeichnis stehen kann.

Beim Einsatz der Komponente ist allerdings das Folgende zu beachten:

- Es wird nicht spezifiziert, wie lange das Auftreten eines Dateisystem-Ereignisses und das Auftreten des entsprechenden Gambas-Ereignisses auseinander liegen können.
- Wenn beispielsweise eine Datei erstellt wird, kann der Gambas-Prozess a priori erst eine Stunde später davon erfahren. Zu diesem Zeitpunkt kann die Datei aber schon wieder gelöscht worden sein. Das liegt in der Natur der Sache, denn die Dateisystem-Ereignisse sind asynchron.
- Unter normaler Betriebslast auf dem Rechner ist der Kernel aber bemüht, die Ereignisse zeitnah zu versenden.
- Ein Gambas-Programm, das die einzige Klasse *Watch* der Komponente gb.inotify einsetzt, wird durch den Linux-Kernel über Aktivitäten im überwachten Verzeichnis informiert, in dem Gambas-Ereignisse des Watch-Objekts ausgewertet werden.

6.8.1 Klasse Watch

Die Klasse Watch (gb.inotify) repräsentiert ein zu überwachendes Dateisystem-Objekt. Sie können die Klasse erzeugen. Die Klasse verhält sich wie ein statisches Array, dessen Werte nur gelesen werden können. Das Beobachten eines Verzeichnisses erfolgt nicht rekursiv für seine Unterverzeichnisse. Das bedeutet, dass Sie nur Ereignisse aus dem Verzeichnis selbst und für seine unmittelbaren Einträge erhalten. Man kann aber für jedes Unterverzeichnis eines Basisverzeichnisses ein separates Watch-Objekt anlegen und mit dem Create-Ereignis sogar zur Laufzeit erzeugte Unterverzeichnisse überwachen.

6.8.2 Erzeugen eines Watch-Objekts

Sie können Watch-Objekte regulär über die *New*-Instruktion erstellen. Die Signatur des Konstruktors ist folgende:

```
hWatch = New Watch ( Path As String [ , NoFollowLink As Boolean, Events As Integer ] ) As "EventName"
```

Zuerst wird der Pfad des zu überwachenden Dateisystem-Eintrags angegeben. Mit dem optionalen Parameter *NoFollowLink* können Sie das Verfolgen symbolischer Links bei der Interpretation Ihres Pfades verbieten. Standardmäßig werden Links verfolgt. Das letzte optionale Argument *Events* ist eine Bitmaske von zu überwachenden Ereignissen. Benutzen Sie hierfür die Konstanten der Watch-Klasse. Wenn Sie zum Beispiel die Create- und Delete-Ereignisse eines Pfades *sPath* überwachen wollen, dann erzeugen Sie das Watch-Objekt so:

```
hWatch = New Watch(sPath, False, Watch.Create + Watch.Delete) As „MyWatch”
```

Geben Sie die Events-Maske nicht an, ermittelt die Watch-Klasse selbstständig alle Events, für die Sie Event-Handler unter dem angegebenen Event-Namen angelegt haben. Haben Sie

```
Private $hWatch As Watch

Public Sub Form_Open()
  ...
  $hWatch = New Watch(sPath) As "MyWatch"
End

Public Sub MyWatch_Create()
  ...
End

Public Sub MyWatch_Delete()
  ...
End
```

deklariert, so erkennt die Watch-Klasse, dass Sie Create- und Delete-Ereignisse von MyWatch abfan-

gen möchten und stellt diese – und nur diese – Ereignisse auf empfangbar. Sie sollten aus Performanzgründen nie mehr Ereignisse in der Events-Maske angeben als Sie benötigen. Über die Events-Eigenschaft der Watch-Klasse können Sie später die empfangbaren Events verändern. Sie verwenden diese Eigenschaft wie ein Array von Boolean-Werten und indizieren Sie mittels der Watch-Konstanten:

```
$hWatch.Events[Watch.Create] = False ' Create-Events sind nun nicht mehr empfangbar
$hWatch.Events[Watch.Move] = True ' Stattdessen interessiert nun das Move-Event
```

Beachten Sie, dass die einzige Aufgabe von Watch-Objekten das Auslösen von Events ist! Wenn Sie vergessen, einem Watch-Objekt einen Event-Namen zu geben, kann es keine Events auslösen und ist nutzlos.

6.8.2.1 Statische Eigenschaften

Die Klasse *Watch* besitzt vier statische Eigenschaften, die Zusatzinformationen für Event-Handler liefern und nur in solchen verwendet werden sollten. Die statischen Eigenschaften werden verwendet, um Daten aus dem Kernel während der Ereignisbehandlungsroutinen zu speichern.

Eigenschaft	Datentyp	Beschreibung
Cookie	Integer	Ein Cookie wird verwendet, um Ereignisse zu verknüpfen. Das ist gegenwärtig (12. Juli 2018) nur für die Verknüpfung von MoveFrom- und MoveTo-Ereignissen der selben Datei nötig.
IsDir	Boolean	Gibt an, ob sich die Beobachtung auf ein Verzeichnis bezieht oder bezog.
Name	String	Gilt nur für überwachte Verzeichnisse: Zurückgegeben wird der Name der Datei oder des Unterverzeichnisses, von dem das Ereignis ausging. Der Name ist relativ zum überwachten Verzeichnis. Wenn das Verzeichnis selbst das Ereignis ausgelöst hat, ist Name = Null. Wird beispielsweise das Verzeichnis "abc" überwacht und dort eine Datei mit dem Dateinamen "xyz" erzeugt, wird das Create-Event von "abc" ausgelöst und Name auf "xyz" gesetzt.
Unmount	Boolean	Gibt zurück, ob das Dateisystem ausgehängt wurde, in dem der beobachtete Pfad lag. In diesem Fall wird das Beobachtungsobjekt unmittelbar nach dem Auslösen des Ereignisses für ungültig erklärt.

Tabelle 6.8.2.1.1 : Statische Eigenschaften der Klasse Watch

6.8.2.2 Eigenschaften

Eigenschaft	Datentyp	Beschreibung
Events	.Watch.Events	Gibt eine virtuelle Klasse zurück, um die Bitmaske der Ereignisüberwachung anzugeben. Diese virtuelle Klasse wird verwendet, um zu verwalten, welche Ereignisse von einem bestimmten Watch-Objekt überwacht werden.
IsPaused	Boolean	Gibt zurück, ob der Watch momentan pausiert (Methoden Pause und Resume).
Path	String	Gibt den überwachten Pfad zurück.
Tag	Variant	Den Einsatz dieser Eigenschaft kann der Gambas-Programmierer frei bestimmen.

Tabelle 6.8.2.2.1 : Eigenschaften der Klasse Watch

6.8.2.3 Methoden

Die Klasse *Watch* verfügt nur über diese zwei Methoden:

Methode	Beschreibung
Pause	Lässt ein Watch-Objekt pausieren und hindert es so daran, Ereignisse auszulösen.
Resume	Beendet den Pause-Modus eines Watch-Objekts.

Tabelle 6.8.2.3.1 : Methoden der Klasse Watch

6.8.2.4 Ereignisse

Die Klasse *Watch* verfügt über diese Ereignisse:

Ereignis	Beschreibung
Close	Das Ereignis wird ausgelöst, wenn die überwachte Datei oder eine Datei im überwachten Verzeichnis geschlossen wird.
Create	Das Ereignis wird ausgelöst, wenn ein Eintrag (Datei oder Verzeichnis) im überwachten Verzeichnis erzeugt wird.
Delete	Das Ereignis wird ausgelöst, wenn ein Eintrag im überwachten Verzeichnis oder das überwachte Objekt selbst gelöscht wurde.
Move	Das Ereignis wird ausgelöst, wenn das überwachte Objekt verschoben wurde.
MoveFrom	Das Ereignis wird ausgelöst, wenn ein Eintrag des überwachten Verzeichnisses verschoben wird (Ereignis für das Quellverzeichnis der Verschiebung).
MoveTo	Das Ereignis wird ausgelöst, wenn ein Eintrag in das überwachte Verzeichnis verschoben wird (Ereignis für das Zielverzeichnis einer Verschiebung).
Open	Das Ereignis wird ausgelöst, wenn die überwachte Datei oder ein Eintrag im überwachten Verzeichnis geöffnet wird.
Read	Das Ereignis wird ausgelöst, wenn auf die überwachte Datei oder einen Eintrag im überwachten Verzeichnis lesend zugegriffen wurde, was u.a. auch das Ausführen einschließt.
Stat	Das Ereignis wird ausgelöst, wenn Meta-Daten oder Datei-Attribute des überwachten Objekts verändert wurden.
Write	Das Ereignis wird ausgelöst, wenn auf die überwachte Datei oder einen Eintrag im überwachten Verzeichnis schreibend zugegriffen wurde.

Tabelle 6.8.2.4.1 : Ereignisse der Klasse Watch

Wie Sie sehen, hängt die Bedeutung und Interpretation eines Ereignisses davon ab, ob eine Datei oder ein Verzeichnis überwacht wird.

6.8.3 Projekt

Das Projekt demonstriert die Überwachung von ausgewählten temporären Verzeichnissen und einer temporären Datei. Die Datei und die Verzeichnisse werden zur Laufzeit angelegt und auf unterschiedliche Weise bearbeitet (öffnen, geändert, verschoben, gelöscht).

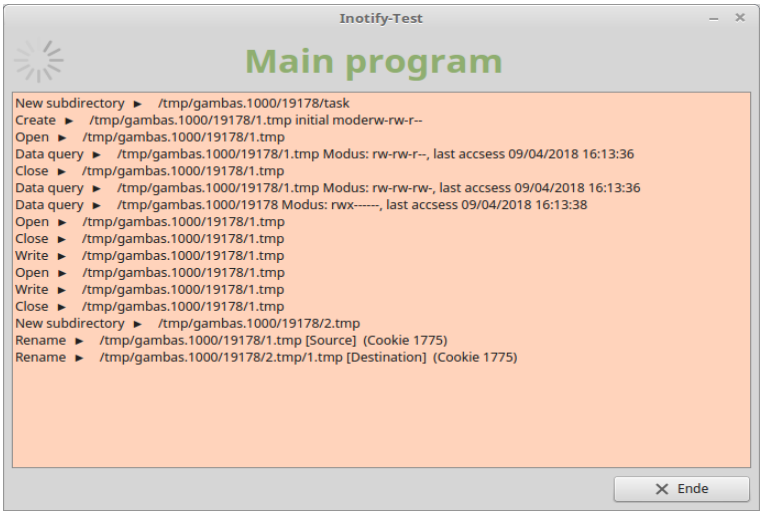

Abbildung 6.8.3.1: Überwachungsprotokoll

Das Besondere an diesem Projekt ist das Auslösen von Ereignissen durch ein externes Skript. Es muss über einen Task realisiert werden, denn einerseits soll die Überwachung möglichst zeitnah erfolgen und andererseits ist der Task im Projekt dazu da, um für Ereignisse zu sorgen, damit man auto-

matisch etwas zu sehen bekommt, wenn man das Projekt startet (Task-Objekt im Kapitel 20.6.0). Würde das Skript synchron ausgeführt, dann würden erst alle Bearbeitungen der Verzeichnisse und der Datei erfolgen und später erst alle Überwachungsergebnisse erhalten. Im Quelltext erkennen Sie auch, dass zu jedem neuen Verzeichnis ein eigenes Watch-Objekt erzeugt wird und in ein Watch[ ]-Array eingefügt wird. Der entsprechende Eintrag im Array $aSubDirs vom Typ Watch[ ] wird entfernt, wenn das zu beobachtende Objekt gelöscht wurde. Wenn man – ohne Task – die Ereignisse erst erhalten würde, nachdem das gesamte Skript durchgelaufen ist, wäre das erstellte Unterverzeichnis bereits gelöscht, wenn der Gambas-Prozess sein Create-Ereignis empfängt. Es kann also nicht mehr überwacht werden und der Prozess erhält insbesondere nicht die (in der Vergangenheit liegenden) Ereignisse, die sich unterhalb dieses Unterverzeichnisses ereignet haben.

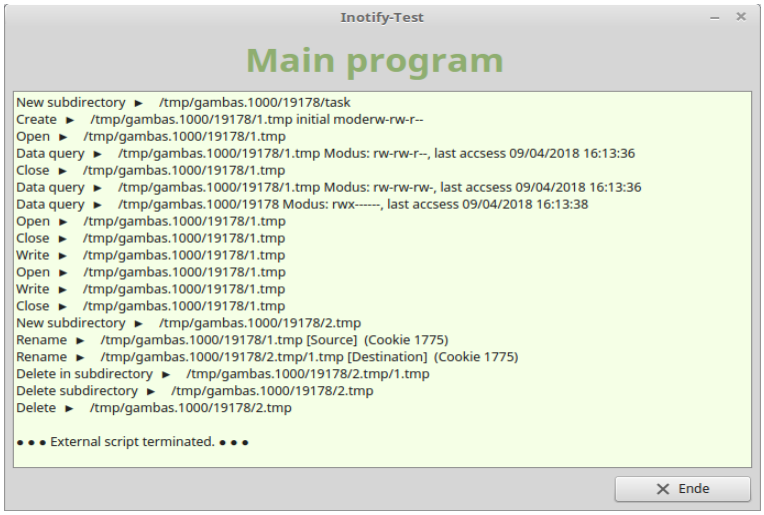

Abbildung 6.8.3.2: Überwachungsprotokoll – Ende

Die Bearbeitung der Verzeichnisse und der Datei leistet die eigene Klasse in *ExternalScript.class*.

Die Quelltexte für das vorgestellte Projekt werden komplett angegeben. Es folgt der Quelltext der Klasse *ExternalScript.class*:

```
' Gambas class file

Inherits Task

Private $sFile As String
Private $sDir As String

Public Sub _new(sFile As String, sDir As String)
  $sFile = sFile
  $sDir = sDir
End

Public Sub Main()
  Shell Subst$("touch &1;  sleep 1; chmod a+w &1; sleep 1;"
  "touch &3;  sleep 1; cat &1; sleep 1;"
  "echo 'test' > &1; sleep 2;"
  "mkdir &2; sleep 1; mv &1 &2; sleep 1;"
  "rm &4; sleep 1; rmdir &2; sleep 1", $sFile, $sDir, File.Dir($sFile), $sDir &/ File.Name($sFile)) Wait
  Print "Finished"
  Flush
  Do ' Wait for the main process to finish the task
    Wait 1
  Loop
End
```

Quelltext *FMain.class*:

```
' Gambas class file

Private $hTmp As Watch
Private $aSubdirs As New Watch[]
Private $hScript As ExternalScript

Public Sub Form_Open()
```

```
  Dim sFile As String = Temp$()
  Dim sDir As String = Temp$()

  FMain.Resizable = False
  TextArea1.ReadOnly = True
  $hTmp = New Watch(File.Dir(sFile)) As "Tmp"
' The script must be a task because the events are as close as possible to each other.
' at the time they are triggered. If the script would be executed in this process,
' you received the events after the complete script has run. So could For example,
' you cannot intercept events in the subdirectories created at runtime.
  $hScript = New ExternalScript(sFile, sDir) As "ExternalScript"
  Spinner1.Start()
End

Public Sub ExternalScript_Read(Data As String)
  If Trim$(Data) <> "Finished" Then Return
  $hScript.Stop() ' Stops the task as a background process
  TextArea1.Insert(gb.NewLine & "● ● ● External script terminated. ● ● ●")
  TextArea1.Background = &HF5FFE6
  Spinner1.Stop()
  Spinner1.Visible = False
End

Public Sub Tmp_Read()
  Note("Lesen")
End

Public Sub Tmp_Create()

  Dim hSubdir As Watch

  If Watch.IsDir Then
' The following try is necessary here to intercept a 'race condition':
' The Subdirectory could have been created and deleted before this
' event handler has been called. The New Watch(..) could fail because
' edit a Create event whose subject has already been deleted.
     Try hSubdir = New Watch(Last.Path &/ Watch.Name) As "Subdir"
     If Not Error Then
        Note("New subdirectory")
        $aSubdirs.Add(hSubdir)
     Endif
  Else
     Note("Create", "initial mode" & Stat(Last.Path &/ Watch.Name).Auth)
  Endif
End

Public Sub Tmp_Open()
  Note("Open")
End

Public Sub Tmp_Close()
  Note("Close")
End

Public Sub Tmp_Write()
  Note("Write")
End

Public Sub Tmp_Move()
  Note("Rename")
End

Public Sub Tmp_MoveFrom()
  Note("Rename", "[Source]", Subst$(" (Cookie &1)", Watch.Cookie))
End

Public Sub Subdir_MoveTo()
  Note("Rename", "[Destination]", Subst$(" (Cookie &1)", Watch.Cookie))
End

Public Sub Tmp_Delete()
  Note("Delete")
End

Public Sub Subdir_Delete()
  Note(Subst$("Delete &1subdirectory", IIf(Watch.Name, "in ", "")))
  If Not Watch.Name Then $aSubdirs.Remove($aSubdirs.Find(Last))
End

Public Sub Tmp_Stat()
  With Stat(Last.Path &/ Watch.Name)
    Note("Data query", Subst$("Modus &1, last accsess &2", .Auth, .LastAccess))
  End With
  Catch
```

```
' Last.Path &/ Watch.Name might no longer exist and Stat might fail.
End

' The three dots indicate that the Note() procedure has a variable argument list.
' After the mandatory parameter sWhat, k >= 0 other arguments of any type can follow.
' Read more about this at http://gambaswiki.org/wiki/lang/methoddecl and
' http://gambaswiki.org/wiki/comp/gb/param
Private Sub Note(sWhat As String, ...)

  Dim sArg As String

  TextArea1.Insert(sWhat & "  ►    " & Last.Path &/ Watch.Name)
  For Each sArg In Param
    TextArea1.Insert(" " & sArg)
  Next
  TextArea1.Insert(gb.NewLine)
  TextArea1.Pos = Len(TextArea1.Text)
  Print Param.Count
End

Public Sub btnClose_Click()
  FMain.Close
End

Public Sub Form_Close()
  If $hScript.Running Then $hScript.Stop
  $hTmp = Null
  $aSubdirs.Clear()
  Wait 1
End
```

Alle überwachten Ereignisse werden angezeigt, wenn diese durch das Skript ausgelöst werden, wie Sie es in der Abbildung 6.8.3.2 im Überwachungsprotokoll sehen können.