10.5.9 Application.Busy

Über den Wert der Eigenschaft *Application.Busy* der Klasse *Application (gb.qt4)* ändern Sie den Beschäftigt-Zustand einer Anwendung.

10.5.9.1 Hinweise

- Die Eigenschaft *Application.Busy* hat den Datentyp *Integer* mit dem Default-Wert 0.
- Sie ändern die Eigenschaft Application.Busy entweder mit 'Inc Application.Busy' und 'Dec Application.Busy' oder mit 'Application.Busy = 1' und 'Application.Busy = 0'. Relevant sind nur die Werte 0 und ein Wert größer als Null.
- Sie sollten den Wert dieser Eigenschaft auf 1 setzen, wenn in der Anwendung eine Anweisung oder eine Folge von Anweisungen gestartet wird, die längere Zeit in Anspruch nimmt oder frei von Unterbrechungen ausgeführt werden soll. Vermindern Sie danach den Wert wieder auf 0.
- Wenn der Wert der Eigenschaft Application.Busy größer als Null ist, wird der Mauszeiger automatisch, aber temporär auf *Mouse.Wait* geändert und damit angezeigt, dass die Anwendung ausgelastet ist und nicht auf Benutzer-Eingaben reagiert.
- Sollte nach der Anweisung 'Inc Application.Busy' oder 'Application.Busy = 1' ein Fehler auftreten, so müssen Sie diesen abfangen und in geeigneter Weise darauf reagieren. Vergessen Sie nicht, den Wert von *Application.Busy* wieder auf 0 zu setzen.

Vorschlag: Spendieren Sie einem Formular in einem Beispiel-Projekt einen Button und fügen Sie den folgenden Quelltext als Ereignisbehandlungsroutine ein, um sich von der Wirkung der Verwendung der Eigenschaft *Application.Busy* zu überzeugen:

```
Public Sub btnSetBusy_Click()
  Dim iCount As Integer

    Inc Application.Busy ' Alternative: Application.Busy = 1
        For iCount = 1 To 40000
            Print Sqr(iCount + Pi(0.533))
        Next ' iCount
    Dec Application.Busy ' Application.Busy = 0

    Message.Info("Das war es auch schon ...")

End ' btnSetBusy_Click()
```

10.5.9.2 Beispiele

Die Beispiele dienen der Demonstration des Einsatzes des Kontroll-Elements *Application.Busy* in unterschiedlicher Verwendung.

Beispiel 1

In einer Applikation für die Installation von Gambas über SVN wird zuerst die lokale Versionsnummer auf dem PC abgefragt und dann die Nummer der aktuellen Revision auf dem SVN-Server. Für diese Abfrage wird das Programm nach außen über den Mauszeiger den "Bin-beschäftigt-Zustand" signalisieren und nicht auf Benutzereingaben reagieren. Tritt ein Fehler auf, wird dieser abgefangen, eine Fehlermeldung ausgegeben und danach das Programm mit 'Dec Application.Busy' wieder freigegeben. Die Freigabe erfolgt auch dann, wenn die Revisionsnummer fehlerfrei ermittelt werden konnte.

```
Public Sub btnUpdateStart_Click()
  Dim sResult, sMessage As String
  Dim iTimeOut As Integer
  ...
  Inc Application.Busy ' Mouse.Wait
    Repeat
       Shell "cd " & sSVNPfad & "; svn info -r HEAD | awk '$1 ~ /^Revision:$/ {print $2;}'" To sResult
       Inc iTimeOut
    Until (sResult <> Null Or iTimeOut > 9)
    Try iSVNCurrentRevision = Val(sResult)
    If Error Then
       Message.Error("Der SVN-Server ist gegenwärtig nicht erreichbar!")
       Dec Application.Busy
       Return
    Endif ' ERROR ?
  Dec Application.Busy ' Mouse.Default
End ' btnUpdateStart_Click()
```

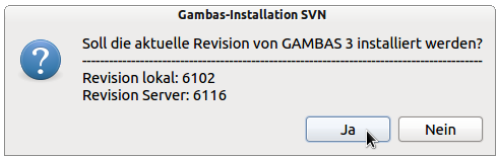

Abbildung 10.5.9.2.1: Die Abfragen waren erfolgreich

Beispiel 2

Ein Druckprogramm nimmt in der Zeit, in der gedruckt wird, keine Eingaben mehr an:

```
Public Sub btnPrintImage_Click()

  If printerImage.Configure() Then Return

  Me.Enabled = False          ' Das Formular wird deaktiviert
     Application.Busy = 1     ' Das Programm nimmt keine Eingaben mehr entgegen ...
         printerImage.Print   ' Der Druck wird gestartet ...
     Application.Busy = 0     ' Das Programm nimmt wieder Eingaben entgegen...
  Me.Enabled = True           ' Das Formular wird aktiviert

End ' btnPrintImage_Click()
```

Beispiel 3

Als Teil-Prozess für die Installation eines Programms (Update) aus vorliegenden Paketen wird die Anweisung *./configure -C* ausgeführt, die das System analysiert, die Pakete konfiguriert sowie das Makefile generiert. Für die Laufzeit des Teil-Prozesses werden vom Haupt-Programm alle Eingaben ignoriert:

```
Public Sub btnConfigureUpdate_Click()

  Inc Application.Busy
    sShellCommand = "echo 'CONFIGURE';echo '---------------------------';"
    sShellCommand &= " cd " & sSVNPfad & ";./configure -C"
    GoToTerminal(GetTerminalList()[0], sShellCommand)
    btnMakeUpdate.Enabled = True
  Dec Application.Busy

End ' btnConfigureUpdate_Click()
```

Beispiel 4

Für das Demonstrationsprogramm zur Klasse *Watcher* (→ Kapitel 20.10 Watcher) sollten die überwachten Ereignisse behandelt und mit Meldungen in eigenen Fenstern dokumentiert werden. Der folgende Quelltext-Ausschnitt aus der Erprobungsphase für das Projekt löste nicht nur Fehler, sondern auch einen Programm-Absturz aus, weil durch das ständige Ändern (Move oder Resize) auch stetig Meldungsfenster erzeugt wurden:

```
Public Sub wWatcher_Resize()
  Message.Info(("Das Hauptfenster " & wWatcherObject1.Control.Name & " hat seine Größe verändert!"))
  Stop Event
End ' wWatcher_Resize()

Public Sub wWatcher_Move()
  Message.Info(("Das Hauptfenster " & wWatcherObject1.Control.Name & " hat seine Position geändert!"))
  Stop Event
End ' wWatcher_Move()
```

Die Überlegungen und Ansätze zur Lösung:

- *Message* ist eine statische Klasse, von der immer höchstens ein Objekt existiert (auch wenn es nicht angezeigt wird).
- Die Event-Kaskade beim manuellen Bewegen/Vergrößern des Fensters kommt so zustande: Sobald die Message-Box des ersten Event-Handlers angezeigt wird, wartet der Interpreter auf das Schließen dieses Dialogs. Das Warten übernimmt der Event-Loop (er wartet darauf, dass der Button im Dialog geklickt wird). Aber dann kommt das nächste Event vom Watcher und der

Interpreter führt den Event-Handler aus, der den Fehler verursacht.

- Eine Lösung zur Vermeidung einer Event-Kaskade besteht darin, den Watcher von der Generierung neuer Events zum Beispiel mit der "Application.Busy"-Strategie abzuhalten!
- Sobald eines der fehlerträchtigen Ereignisse (Move oder Resize) ausgelöst wird, wird im Event-Handle die Eigenschaft *Application.Busy* auf 1 gesetzt und die Message-Box aufgerufen. Nach dem schließen der Box, wird die Eigenschaft wieder auf 0 gesetzt und das Ereignis gestoppt.
- Nur wenn die Eigenschaft *Application.Busy* den Wert 1 hat, ruft der zuständige Event-Handler die Message-Box auf.

Hier der korrigierte Quelltext-Ausschnitt:

```
Private sControlName As String
…
sControlName = "'" & wWatcherObject1.Control.Name & "'"
…
Public Sub wWatcher_Resize()
  Application.Busy = 1
   If Application.Busy = 1 Then Message.Info("Das Fenster " & sControlName & " hat die Größe verändert!")
  Application.Busy = 0
  Stop Event
End ' wWatcher_Resize()

Public Sub wWatcher_Move()
  Inc Application.Busy
   If Application.Busy = 1 Then Message.Info("Das Fenster " & sControlName & " hat die Position geändert!")
  Dec Application.Busy
  Stop Event
End ' wWatcher_Move()
```