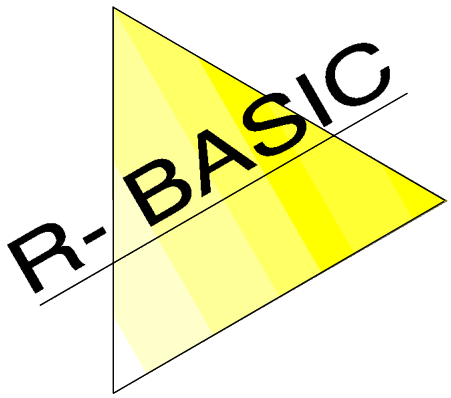
R-BASIC

Einfach unter PC/GEOS programmieren



Objekt-Handbuch

Volume 3
Application, Primary, Button, Group, Menu

Version 1.0

(Leerseite)

Inhaltsverzeichnis

(Leerseite)

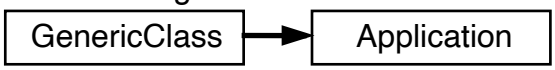
4 Verfügbare Generic Objekt Klassen

In diesem Abschnitt werden die einzelnen generischen Objektklassen, die in R-BASIC verfügbar sind, im Detail beschrieben. Einige dieser Objekte sind sehr einfach zu benutzen, können aber trotzdem sehr komplexe Funktionen ausführen. Da alle diese Objekte von der GenericClass abstammen erben sie sämtliche in den vorherigen Kapiteln besprochenen Instance-Variablen. Sie sollten außerdem die in Kapitel 2 beschriebenen Konzepte kennen, um die damit verbundenen Details zu verstehen.

4.1 Application

Dieses Objekt stellt die Verbindung zum GEOS-System her und ist das top Parent-Objekt für alle anderen generischen BASIC-Objekte ihres Programms. Jedes BASIC-Programm muss genau ein Application-Objekt haben. Direktes Child des Application-Objekts ist im allgemeinen ein Primary-Objekt.

Abstammung:



Die Definition eines einfachen Application-Objekts sieht z.B. so aus

```
Application  MyApp
  Children = MyPrimary
END OBJECT
```

MyApp ist der Name des Application-Objekts, unter dem es bei Bedarf im BASIC-Code angesprochen werden kann. Das ist für Application Objekte jedoch nur selten nötig.

Das Application-Objekt selbst erscheint nicht auf dem Bildschirm. Es ergibt daher keinen Sinn, ihm eine Caption$ zugeben, auch wenn das syntaktisch möglich ist.

Tastaturhandling

Sie können sich in das Tastaturhandling einklinken, indem Sie einen Tastaturhandler für das Application-Objekt schreiben. Dazu werden die folgenden Instancevariablen unterstützt:

Actionhandler	Instancevariablen	Methoden
OnKeyPresed	inputFlags	—

Der OnKeyPressed-Handler muss als KeyboardAction deklariert werden. Eine ausführliche Beschreibung, wie man einen Tastaturhandler schreibt und was es dabei zu beachten gilt, finden Sie im Handbuch "Spezielle Themen", Kapitel 14.

Das Schreiben eines Tastaturhandlers für das Application-Objekt ist eine sehr einfache Methode, wenn ein "globaler" Tastaturhandler geeignet ist. Das ist z.B. für viele Spielprogramme der Fall.

Focus und Target

Das Application-Objekt ist ein Knoten in der Focus- und Target-Hierarchie. Es ist möglich zu überwachen, ob ein Application-Objekt den Focus oder das Target hat, indem man einen Focus- bzw. Target-Handler schreibt. Die notwendigen Details zur Arbeit mit Focus und Target finden Sie im Kapitel 12 (Focus und Target) des Handbuchs "Spezielle Themen". Das Arbeiten mit Focus und Target ist etwas für erfahrene Programmierer und nur in wenigen Fällen notwendig. Eine Ausnahme bildet die Implementation von speziellen Menüs wie dem "Bearbeiten" Menü. Diesem Thema ist deswegen ein eigenes Kapitel ("Spezielle Themen", Kapitel 13) gewidmet.

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
OnInit	OnInit = **<Handler>**	—
OnStartup	OnStartup = **<Handler>**	—
OnExit	OnExit = **<Handler>**	—
OnClpChange	OnClpChange = **<Handler>**	nur schreiben
OnConnection	OnConnection = **<Handler>**	nur schreiben
LongName$	LongName$ = **"Name"**	—
userNotes$	userNotes$ = **"Text"**	—
crNote$	crNote$ = **"Text"**	—
AppToken	AppToken = **"<chars>",<numVal>**	—
DocToken	DocToken = **"<chars>",<numVal>**	—
ExtraToken	ExtraToken = **"<chars>",<numVal>**	—

Methoden:

Methode	Aufgabe
MarkBusy	Busy-Status aktivieren
MarkNotBusy	Busy-Status verlassen
HoldUpInput	Usereingaben zwischenspeichern
ResumeInput	Zwischengespeicherte Usereingaben ausführen
IgnoreInput	Alle Usereingaben ignorieren
AcceptInput	Usereingaben wieder akzeptieren

Action-Handler-Typen:

Handler-Typ	Parameter
SystemAction	(sender as object, flags as word, dataFile$ as String(250))

Konstanten für Parameter "flags"

Konstante	Wert	Bedeutung
AF_FOR_PRINT	1	Datei zum Drucken übergeben
AF_RESTORE	2	Wiederherstellung nach System Shutdown
AF_DATA_FILE	4	Datendatei wurde übergeben
AF_SHUTDOWN	8	GEOS fährt herunter

Eine ausführliche Beschreibung der einzelnen Flags finden Sie im nächsten Kapitel.

4.1.1 LongName, Benutzer Notizen und Tokens

Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
LongName$	LongName$ = **"Name"**	—
userNotes$	userNotes$ = **"Text"**	—
crNote$	crNote$ = **"Text"**	—
AppToken	AppToken = **"<chars>",<numVar>**	—
DocToken	DocToken = **"<chars>",<numVar>**	—
ExtraToken	ExtraToken = **"<chars>",<numVar>**	—

LongName$

Der "**LongName$**" ist der eigentliche Name des Programms, mit dem es im GeoManager und im Expressmenü erscheint. Er wird benötigt, wenn Sie ein "Eigenständiges Programm" anlegen, um es an andere Nutzer weiterzugeben. In den meisten Fällen wird die Instancevariable **LongName$** jedoch gar nicht explizit gesetzt. R-BASIC verwendet dafür standardmäßig den Namen Ihres BASIC Programms. Setzen Sie einen abweichenden Wert für "**LongName$**", falls das fertige Programm unter einem anderen Namen im GeoManager erscheinen soll.

userNotes$

Die **userNotes$** sind die Dokument-Notizen, die Im GeoManager angesehen und geändert werden können. Sie können bis zu 99 Zeichen lang sein, wobei Zeilenumbrüche als ein Zeichen gezählt werden.

crNote$

crNote$ definiert eine bis zu 32 Zeichen lange Copyright-Notiz. Sie wird in den Dateiheader des Programms geschrieben und nicht im GeoManager angezeigt.

AppToken

AppToken spezifiziert das Icon (Symbol-Bild), dass der GeoManager für das Programm anzeigt. Es wird benötigt, wenn Sie ein "Eigenständiges Programm" anlegen, um es an andere Nutzer weiterzugeben. **AppToken** ist vom Typ GeodeToken, d.h. es besteht aus 4 Buchstaben und einer Zahl zwischen 0 und 65535, der sogenannten "Manufacturer-ID".

Beispiel:

```
Application  MyApp
  Children = MyPrimary
  AppToken = "BPdm",16600
  userNotes$ = "Version 2.0, Jan. 2013"
  crNote$ = "(c) by Rabe-Soft 01/2013"
END OBJECT
```

Jedes Programm benötigt ein eigenes Token. Wenn Sie ein "eigenständiges Programm" erzeugen (also ein Programm, das auch ohne R-BASIC lauffähig ist), kopiert R-BASIC das Icon-Bild aus der Token Database in das fertige BASIC Programm. Das bedeutet folgendes:

- Dass durch **AppToken** spezifizierte Token muss sich beim Anlegen des eigenständigen Programms in ihrer Token Database befinden.
- Benutzt ein User Ihr Programm, so installiert das Programm automatisch sein **AppToken** in der Token Database, falls es dort noch nicht existiert. Hier verhält sich ein R-BASIC Programm genau wie ein mit dem PC/GEOS-SDK geschriebenes Programm.
- Spezifizieren Sie kein **AppToken** verwendet R-BASIC ein "Standard-Token".

Sie können für **AppToken** sowohl ein Token aus einer existierenden Iconsammlung verwenden als auch ein eigenes Icon erstellen.
Um ein eigenes Icon zu erstellen können Sie das Programm "IconEditor" verwenden, der das Icon direkt in die Token Database exportieren kann. Dazu benötigen Sie eine eigene Manufacturer-ID. Im Kapitel 3.4 "Über die Manufacturer-ID" im R-BASIC Benutzer Handbuch wird beschrieben, wie Sie eine Manufacturer-ID erhalten können.

Zu einem "Token" (4 Zeichen + Manufacturer-ID) gehören üblicherweise mehrere Icon-Bilder. R-BASIC unterstützt genau zwei Bilder pro Token:
Das Erste ist das eigentliche Icon. Es wird vom GeoManager bei der Anzeige des Programms verwendet. Es ist üblicher Weise 48x30 Pixel groß und hat 16 oder 256 Farben. Das Zweite ist das "Tool" Icon und wird z.B. für das Expressmenü benutzt. Es ist üblicherweise 15x15 Pixel groß und kann ebenfalls 16 oder 265 Farben haben. Im Abschnitt "Eigenständige Programme anlegen" des R-BASIC Benutzer Handbuchs finden sie eine ausführliche Beschreibung wie Sie zum Erstellen eigener Icons vorgehen können.

DocToken

Wenn Ihr Programm mit Dokumenten arbeitet, können Sie diesen ein eigenes Token zuweisen (bei DOS-Dateien in der GEOS.INI, bei VM-Dateien über das Attribut "Token"). Um sicherzustellen, dass sich das Token auch in der Token Database befindet, können Sie die Instancevariable **DocToken** verwenden. R-BASIC wird - wie beim AppToken - beim Anlegen des eigenständigen Programms die beiden zum **DocToken** gehörenden Iconbilder in das BASIC-Programm kopieren. Das BASIC-Programm installiert dann das **DocToken** gemeinsam mit dem AppToken in der Token Database. Mehr macht das R-BASIC Programm nicht! Es bleibt Ihnen überlassen ob und wie Sie das **DocToken** verwenden.

ExtraToken

Für den unwahrscheinlichen Fall, dass Sie ein weiteres Token benötigen, können Sie ein **ExtraToken** spezifizieren, dass dann gemeinsam mit dem AppToken und dem DocToken in das BASIC-Programm kopiert und von diesem bei Bedarf in die Token Database installiert wird.

4.1.2 Actionhandler des Application Objekts

Alle Actionhandler des Application-Objekts müssen als **SystemAction** deklariert sein. Der Parameter **sender** enthält dabei immer das Application Objekt, die Parameter **flags** und **dataFile$** sind im Folgenden beschrieben.

Der Parameter **flags**

Der Parameter "flags" (engl. Flagge) enthält zusätzliche Informationen, die unter gewissen Umständen von Bedeutung sein könnten. Es handelt sich dabei um einzelne Bits, die gesetzt sein können oder eben nicht. Um abzufragen, ob ein bestimmtes Flag gesetzt ist verwenden Sie die logische AND Operation und prüfen Sie, ob das Resultat Null ist oder nicht. Vergessen Sie die Klammern nicht!

```
  ! Prüfen ob AF_DATA_FILE gesetzt ist
  IF   flags AND AF_DATA_FILE   THEN ...
' gleichbedeutend mit
  IF   (flags AND AF_DATA_FILE) <> 0  THEN ...
```

oder:

```
  ! Prüfen ob AF_DATA_FILE  NICHT gesetzt ist
  IF   (flags AND AF_DATA_FILE) = 0   THEN ...
```

Beachten Sie folgendes:

- Verwenden Sie **nicht** den **NOT** Operator
  z.B. ~~IF **NOT** (flags AND AF_DATA_FILE) THEN ...~~
  oder ~~IF flags AND **NOT** AF_DATA_FILE) THEN ...~~
  Der Operator NOT führt eine **bitweise** Negation aus, was gemeinsam mit dem AND Operator in den meisten Fällen zur Aussage "wahr" (d.h. ungleich Null) führt. Ein solcher Fehler ist selbst für Profis sehr schwer zu finden.
- Verwenden Sie niemals ein einfaches "=" (if flags=AF_DATA_FILE), da sie nie sicher sein können ob noch andere Flags gesetzt sind! Möglicherweise werden von späteren Versionen zusätzliche Flags bereitgestellt, die gesetzt sein könnten!

AF_DATA_FILE: Dieses Flag zeigt an, dass dem Handler eine Datendatei übergeben wurde (Parameter dataFile$ enthält dann den Pfad zur Datei). Die Abfrage dieses Flags ist schneller als das Prüfen der Stringlänge mit Len(dataFile$). Len(dataFile$) > Null zeigt ebenfalls an, dass eine Datendatei übergeben wurde.

AF_FOR_PRINT: Dieses Flag kann nur gesetzt sein, wenn auch eine Datendatei übergeben wurde (AF_DATA_FILE gesetzt). Es zeigt an, dass der Nutzer die Datei nicht durch einen Doppelklick sondern über das Menü "Drucken" des Geomanagers geöffnet hat. Sie können z.B. entscheiden, dass Sie die Datei ignorieren wollen (weil Ihr Programm gar nicht drucken kann) oder eine andere Sonderaktion durchführen, wenn das sinnvoll ist.

AF_RESTORE: Dieses Flag kann nur im OnInit oder im OnStartup Handler gesetzt sein. Es zeigt an, dass das Programm beim letzten Herunterfahren von PC/GEOS noch offen war und jetzt, beim Systemneustart, wiederhergestellt (engl.: restored) wird. Möglicherweise möchten Sie in diesem

Fall bestimmte Initialisierungsschritte auslassen oder stattdessen andere ausführen.

AF_SHUTDOWN: Dieses Flag wird nur dem OnExit Handler übergeben und zeigt an, dass sich Ihr Programm schließt, weil PC/GEOS heruntergefahren wird, nicht weil das Programm normal beendet wurde. Sie können dann z.B. Werte, die einen System-Restart überleben sollen in eine Datei sichern und beim wieder Hochfahren des Systems (AF_RESTORE im OnStartup bzw. OnInit Handler gesetzt) wieder auslesen. Es ist aber ein besserer Stil (und meist auch einfacher) Ihr Programm gleich so zu schreiben, dass dies nicht nötig ist.

Der Parameter **dataFile$**

Sie können R-BASIC Programme wie alle anderen GEOS Programme auch durch einen entsprechenden Eintrag in der GEOS.INI (filenameTokens) mit DOS Dateien bzw. durch Setzten des "Creator" Attributs in VM-Dateien auch mit VM-Dateien verknüpfen. Der Geomanager stellt beim Doppelklick auf eine so verknüpfte Datei eine Verbindung zum zugehörigen Programm her und übergibt ihm den Namen und den Pfad zur Datei. Der Parameter "dataFile$" enthält dann den vollständigen Pfad zu der an das Programm übergebenen Datendatei (z.B. "D:\GEOS\DOCUMENT\INFO.FOO"). In diesem Fall ist auch immer das Flag AF_DATA_FILE im Parameter "flags" gesetzt. Wurde keine Datendatei übergeben so enthält dataFile$ einen Leerstring und das Flag AF_DATA_FILE ist nicht gesetzt.
Hinweis: Für den Zugriff auf den Inhalt von VM-Dateien müssen Sie die VMFiles Library einbinden.

4.1.3 Starten und Beenden eines Programms

Das Application Objekt stellt nicht nur die Verbindung zum GEOS System her, es erledigt auch alle Aufgaben, die beim Starten und Beenden eines BASIC Programms anfallen. Dazu gehört insbesondere, das Ausführen des Programmcodes zu starten.

R-BASIC erlaubt es, Programmcode zu schreiben, der nicht Teil einer Routine (SUB oder FUNCTION) oder eines Actionhandlers ist. Alle "klassischen" BASIC Programme und auch viele R-BASIC Beispiele machen davon Gebrauch. Dieser sogenannte "klassische Code" wird beim Start des Programms automatisch ausgeführt.
Beispiel: Ein "Hallo Welt" Programm im klassischen BASIC

```
ClassicCode
CLS
Print : Print
Print "Hallo Welt"
Print "Willkommen bei R-BASIC!"
```

Die Anweisung ClassicCode bewirkt drei Dinge:
- Das Scheiben von Code außerhalb von Routinen wird zugelassen.
- Es werden automatisch ein paar Objekte angelegt, so dass Grafik und Text ausgegeben werden können.
- Dieser "klassische" Code wird beim Start des Programms automatisch ausgeführt.

Für ein objektorientiertes Programm ist das nicht nur ein schlechter Stil, sondern es fehlt z.B. die Möglichkeit speziellen Code am Programmende automatisch auszuführen. Das Application Objekt unterstützt deswegen mehrere Actionhandler, die am Programmstart bzw. am Programmende ausgeführt werden.

4.1.3.1 Programmstart

Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
OnInit	OnInit = **<Handler>**	—
OnStartup	OnStartup = **<Handler>**	—

Wenn ein BASIC Programm startet werden die folgenden Schritte in der hier angegebenen Reihenfolge ausgeführt:

1. Die Objekte werden geladen und vollständig initialisiert, erscheinen aber noch nicht auf dem Schirm.
2. Wenn vorhanden wird der **OnInit** Handler ausgeführt. Dieser Handler kann bereits mit den Objekten interagieren und Dinge erledigen, die vor allen anderen erledigt werden müssen.
3. Die Objekte erscheinen auf dem Schirm.
4. Wenn vorhanden wird der **OnClpChange** Handler ausgeführt.
5. Wenn es Objekte gibt, die einen **OnDraw** Handler oder einen **QueryHandler** haben, werden diese Handler jetzt ausgeführt, damit diese Objekte korrekt dargestellt werden.
6. Wenn vorhanden wird der **OnStartup** Handler ausgeführt. Wenn Sie nicht sicher sind, ob eine Aktion in den **OnInit** oder den **OnStartup** Handler gehört, wählen Sie den OnStartup Handler.
7. Wenn vorhanden wird jetzt der "klassische Code" ausgeführt.

Nach diesen Schritten ist das Programm bereit für weitere Ereignisse.

OnStartup

Der **OnStartup** Handler ist der übliche Platz für den Initialisierungscode Ihres Programms. Der oben dargestellte "klassische" Code würde in der (besseren) objektorientierte Version so aussehen. Die erste Codezeile im Starthandler verhindert, dass die Grafikausgabe nach einem Neustart von GEOS mit laufendem BASIC-Programm erneut ausgeführt wird. Sie sollten **immer** im Blick haben, was im OnStartup bzw. OnInit Handler passiert, wenn GEOS bei laufendem Programm neu gestartet wird. Typische Fehler sind hier z.B. das Initialisieren von Objekten mit vorgegebenen Werte - womit bereits von Nutzer veränderte Werte überschrieben werden - oder das erneute Anwenden von Koordinatentransformationen oder grafischen Ausgaben.

UI Code:

```
Application  MyApp
  Children = MyPrimary
  OnStartup = StartHandler
END OBJECT
```

BASIC Code:

```
SystemAction StartHandler
  IF  flags AND AF_RESTORE   THEN   RETURN
  CLS
  Print : Print
  Print "Hallo Welt"
  Print "Willkommen bei R-BASIC!"
  END Action
```

Wenn das R-BASIC Programm mit einem Dateityp verknüpft ist und durch Doppelklick auf eine verknüpfte Datei gestartet wird, dann enthält der Parameter **dataFile$** den vollständigen Pfad zu dieser Datei. Andernfalls enthält dataFile$ einen Leerstring. Details dazu sind im Abschnitt zur Arbeit mit Dokumenten (siehe unten) beschrieben.
Im Parameter **flags** können eines oder mehrere der Flags AF_FOR_PRINT, AF_RESTORE oder AF_DATA_FILE gesetzt sein.

OnInit

In einigen Fällen kann es nötig sein bereits BASIC Code auszuführen, wenn noch kein Objekt auf dem Schirm ist. Typisch ist z.B. das Öffnen von Dateien und das Einlesen von Daten, die für Objekte benötigt werden, die bereits beim Programmstart sichtbar sind. Der **OnInit** Handler ist immer der allererste BASIC Handler, der ausgeführt wird. Beispiel:

UI Code

```
Application  MyApp
  Children = MyPrimary
  OnInit = InitHandler
  OnExit = ExitHandler          ' Siehe Abschnitt Programmende
END OBJECT
```

BASIC Code

```
DIM     f AS FILE             ' globale Variable

SYSTEMACTION InitHandler
DIM anz as word
  f = FileOpen "MyData.TXT"
  IF  f = NullFile() THEN  f = CreateNewDataFile
  anz = FindAnzahlDatenInFile(f)
  MyDynamicList.count = anz
END Action
```

Wir nehmen in diesem Beispiel an, dass die Routinen CreateNewDataFile und FindAnzahlDatenInFile sowie die DynamicList MyDynamicList anderswo im Code definiert sind.

Hinweise:

- Die meisten Programme benötigen keinen **OnInit** Handler.
- Während der **OnInit** Handler ausgeführt wird sind noch keine Objekte auf dem Schirm. Sie erscheinen erst wenn der OnInit Handler abgearbeitet ist. Sehr umfangreiche (lange laufende) **OnInit** Handler verzögern deswegen den "gefühlten" Programmstart
- Prinzipiell gibt es keine Einschränkungen bezüglich der in einem **OnInit** Handler verwendbaren Befehle. Insbesondere ist es zulässig mit Objekten zu interagieren, deren Instancevariablen zu lesen oder zu ändern.
- Der OnInit-Handler wird auch beim Neustart von GEOS bei laufendem BASIC-Programm ausgeführt. Mit der Codezeile

```
IF  flags AND AF_RESTORE THEN RETURN
```

können Sie verhindern, dass in diesem unerwünschter Initialierungscode erneut ausgeführt wird.

- Vermeiden Sie die Verwendung von "klassischen" Interaktionsbefehlen wie INPUT oder InKey$ im **OnInit** Handler. Das erzeugt nur Chaos.
- Wenn Sie nicht sicher sind, ob sie den **OnInit** oder den **OnStartup** Handler verwenden sollen, wählen Sie zunächst den **OnStartup** Handler. Nur wenn Sie mit dem Ergebnis nicht zufrieden sind verschieben Sie Teile des Codes in den **OnInit** Handler.

Wenn das R-BASIC Programm mit einem Dateityp verknüpft ist und durch Doppelklick auf eine verknüpfte Datei gestartet wird, dann enthält der Parameter **dataFile$** den vollständigen Pfad zu dieser Datei. Andernfalls enthält dataFile$ einen Leerstring. Details dazu sind im Abschnitt zur Arbeit mit Dokumenten (siehe unten) beschrieben.
Im Parameter **flags** können eines oder mehrere der Flags AF_FOR_PRINT, AF_RESTORE oder AF_DATA_FILE gesetzt sein.

4.1.3.2 Programmende

Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
OnExit	OnExit = **<Handler>**	—

Ein BASIC Programm kann durch den Menüeintrag "Beende ..." oder durch den BASIC Befehl EXIT beendet werden. Auch im Falle eines Laufzeitfehlers wird das Programm automatisch beendet. In allen diesen Fällen erledigt das Application Objekt die folgenden Schritte:

1. Wenn vorhanden wird der **OnExit** Handler ausgeführt. Sie können hier z.B. abfragen, ob geänderte Daten gespeichert werden sollen.
2. Die Objekte werden vom Schirm genommen, bleiben aber noch intakt.
3. Die Event-Warteschlage wird geleert. Events, die noch in der Warteschlange stehen, werden nicht mehr ausgeführt, d.h. die dazugehörigen Handler werden nicht mehr gerufen.
4. Die Objekte und internen Datenstrukturen werden aufgeräumt.

Hinweise:

- Der **OnExit** Handler (wenn vorhanden) wird auch dann ausgeführt, wenn es vorher einen Laufzeitfehler gegeben hat.
- Beim Herunterfahren von PC/GEOS die globalen BASIC Variablen **nicht** automatisch gesichert. Sie sollten ihre Programme grundsätzlich so schreiben dass dies nicht nötig ist - oder Sie müssen sich selbst darum kümmern.

OnExit

Der **OnExit** Handler wird automatisch ausgeführt, wenn das Programm geschlossen wird. Er muss als **SystemAction** deklariert sein. Sie sollten im **OnExit** Handler alle Ressourcen freigeben (z.B. Dateien schließen) die Sie im OnInit oder im OnStartup Handler angefordert haben.

Beispiel (bezieht sich auf den Code des OnInit Handlers oben):

```
SYSTEMACTION ExitHandler
  FileClose f
  f = NullFile()
END Action
```

Im Parameter **flags** kann das Flag AF_SHUTDOWN gesetzt sein. In diesem Fall darf man keine Messageboxen oder Dialoge aktivieren, sonst hängt das System.

Beispiel 2: Die SUB's DoSaveData (Daten speichern) und DoCloseFile (Datei Schließen) müssen irgendwo anders definiert sein.

```
SYSTEMACTION ExitHandler
  IF (flags AND AF_SHUTDOWN) = 0 THEN
    IF QuestionBox ("Daten speichern?") = YES THEN DoSaveData
  End IF
  DoCloseFile
END Action
```

4.1.4 Arbeit mit Dokumenten

Im Handbuch "Spezielle Themen", Kapitel 15 finden Sie eine ausführliche Beschreibung, wie man ein komplettes Dokument-Interface implementiert.

Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
OnConnection	OnConnection = **<Handler>**	nur schreiben

Sie können R-BASIC Programme durch einen entsprechenden Eintrag in der GEOS.INI (filenameTokens) mit DOS Dateien bzw. durch Setzten des "Creator" Attributs in GEOS-Dateien auch mit GEOS-Dateien verknüpfen. "Startet" der Nutzer eine solche Datei z.B. durch Doppelklick im Geomanager so wird die Datei im Parameter dataFile$ (vgl. Kapitel 4.1.1) je nach Situation an den OnInit bzw. den OnStartup oder an den **OnConnection** Handler übergeben. Gemeinsam mit dem OnStartup Handler (seltener mit dem OnInit Handler) können Sie mit dem **OnConnection** Handler die Arbeit mit Dokumenten organisieren.

OnConnection

Der **OnConnection** Handler wird gerufen, wenn ein anderes Programm (z.B. der Geomanager) einer Verbindung (Connection) zum BASIC Programm herstellt und der OnInit bzw. OnStartup Handler nicht in Frage kommt, da das BASIC Programm bereits läuft. Üblicher Weise wird eine solche Connection hergestellt, wenn das BASIC Programm mit einem Dateityp verknüpft ist und der Nutzer eine so verknüpfte Datei im Geomanager durch Doppelklick startet.

Dabei gelten die folgenden Regeln:

- Wenn ein R-BASIC Programm durch Doppelklick auf eine verknüpfte Datei gestartet wird, so wird der Pfad zu dieser Datei sowohl dem **OnInit** als auch dem **OnStartup** Handler im Parameter **dataFile$** übergeben. **OnConnection** wird beim Öffnen des Programms nicht gerufen!
- Doppelklickt der Nutzer eine verknüpfte Datei jedoch während das R-BASIC Programm läuft wird der **OnConnection** Handler gerufen, wobei auch hier der Parameter **dataFile$** den vollständigen Pfad zur übergebenen Datei (z.B. "D:\GEOS\DOCUMENT\DEMO.RBF") enthält.
- In beiden Fällen ist im Parameter **flags** das Flag AF_DATA_FILE gesetzt, das Flag AF_FOR_PRINT kann zusätzlich gesetzt sein.

Achtung! Es ist denkbar, wenn auch sehr unwahrscheinlich, dass andere Programme (außer dem Geomanager) eine Verbindung zu Ihrem BASIC Programm herstellen. In diesem Fall kann ebenfalls eine Datendatei übergeben werden, häufiger ist jedoch, dass in diesem Fall keine Datei übergeben wird. Prüfen Sie in ihrem **OnConnection** Handler also immer das Bit AF_DATA_FILE (bzw. die Länge von dataFile$) ab!

Beispiel

Nehmen wir an, Sie möchten einen Viewer für R-BASIC Blockgrafik Fontdateien (RBF-Dateien) schreiben. Der Viewer soll beim Doppelklick auf eine RBF-Datei gestartet werden bzw., wenn er bereits läuft, die angeklickte Datei anzeigen. Dazu nehmen wir folgendes als gegeben an:

- Es existiert eine globale Dateivariable f. Diese referenziert die offene Fontdatei.
- Die selbst geschriebene Routine DisplayFontData zeigt die gewünschten Inhalte an.

```
  DECL SUB DisplayFontData(fh as FILE)
```

- Der Geomanager zeigt RBF-Dateien mit dem Token "Font",5 an.
- Ihr Viewer hat das Token "RbfV",16600.

In der GEOS.INI muss sich die folgende Verknüpfung befinden:

```
[fileManager]
filenameTokens = {
    *.RBF="Font",5,"RbfV",16600
  ....
}
```

Ausschnitt aus dem UI Code:

```
Application RBFViewerApplication
  OnStartup = ViewerStartupHandler
  OnConnection = ViewerNewFileHandler
  OnExit = ViewerExitHandler
  AppToken = "RbfV",16600
  DocToken = "Font",5
  ....
End OBJECT
```

Der passende BASIC Code dazu:

```
SYSTEMACTION ViewerStartupHandler
  ! Prüfen ob eine Datei übergeben wurde
  IF Len(dataFile$) THEN
    f = FileOpen ( dataFile$, "r")    ! Nur Lesen reicht.
    DisplayFontData ( f )
    End IF
  ! Alternativ könnte man auch AF_DATA_FILE prüfen
  ! IF  flags AND AF_DATA_FILE  THEN ...
END Action

SYSTEMACTION ViewerNewFileHandler
  ! Prüfen ob überhaupt eine Datei übergeben wurde
  IF (flags AND AF_DATA_FILE) = 0 THEN return
  ! eventuell offene Datei schließen
  IF  f <> NullFile() THEN FileClose(f)
  ! Neue Datei anzeigen
  f = FileOpen ( dataFile$, "r")
  DisplayFontData(f)
```

```
END Action

SYSTEMACTION ViewerExitHandler
  IF  f <> NullFile() THEN FileClose(f)
END Action
```

Bei Bedarf können Sie noch ein Dateimenü (Buttons "Öffnen" und "Schließen") hinzufügen.

4.1.5 Überwachung der Zwischenablage

OnClpChange

Um in R-BASIC z.B. ein "Bearbeiten" Menü zu implementieren müssen Sie wissen, wenn jemand etwas ins Clipboard kopiert und was es ist. Dann können Sie z.B. einen "Einfügen" Schalter enablen oder disablen. Für dieses Zweck verfügt das Application Objekt über einen speziellen Actionhandler, der immer dann aufgerufen wird, wenn sich im Clipboard etwas tut. Der ActionHandler muss als "SystemAction" implementiert sein.

Im UI Code:

```
Application DemoApplication
    Children = DemoPrimary
    OnClpChange = ClpChangeHandler
END Object

' folgende Objekte sollen existieren:
BitmapContent DemoBitmap
Button PasteButton
```

Im BASIC Code:

```
'
' Der Handler enabled oder disabled den Einfügen Button
'
SYSTEMACTION ClpChangeHandler
DIM ok
  ok = DemoBitmap.ClpTestPaste
  IF ok THEN
    PasteButton.enabled = TRUE
    ELSE
    PasteButton.enabled = FALSE
    END IF
END Action
```

Der OnClpChange Handler wird automatisch immer dann aufgerufen, wenn sich die Daten im Clipboard ändern. Die übergebenen Parameter sind hier ohne Bedeutung und sollten ignoriert werden. Details zur Arbeit mit dem Clipboard und dem OnClpChange Handler erfahren Sie im Kapitel "Arbeit mit der Zwischenablage" und insbesondere im Abschnitt "Das Clipboard überwachen"

4.1.6 Der Busy-Status

Gelegentlich kommt es vor, dass Ihr Programm "beschäftigt" ist und nicht sofort auf weitere Usereingaben reagieren kann. Der übliche Weg, dies dem User deutlich machen ist, den Mauszeiger zu einer "Sanduhr" werden zu lassen. Für diesen Zweck gibt es den "Busy" (= beschäftigt) Status. Je nachdem, ob sie während dessen auf die Usereingaben reagieren können oder wollen gibt es verschiede Wege, den Busy-Status zu aktivieren.

Methoden:

Methode	Aufgabe
MarkBusy	Busy-Status aktivieren
MarkNotBusy	Busy-Status verlassen
HoldUpInput	Usereingaben zwischenspeichern
ResumeInput	Zwischengespeicherte Usereingaben ausführen
IgnoreInput	Alle Usereingaben ignorieren
AcceptInput	Usereingaben wieder akzeptieren

Syntax BASIC-Code: **<obj>.MarkBusy**
<obj>.MarkNotBusy

Selten genutzte Methoden:
<obj>.HoldUpInput
<obj>.ResumeInput

<obj>.IgnoreInput
<obj>.AcceptInput

Wichtig! Alle hier aufgeführten Methoden sind kumulativ, das heißt sie können mehrfach hintereinander ausgeführt werden und z.B. zu jedem MarkBusy wird ein eigenes MarkNotBusy benötigt.

```
DemoAppliacation.MarkBusy   ' jetzt busy
DemoAppliacation.MarkBusy   ' 2x busy
DemoAppliacation.MarkNotBusy    ' immer noch busy
DemoAppliacation.MarkNotBusy    ' jetzt nicht mehr
```

MarkBusy, MarkNotBusy

Der Aufruf von MarkBusy lässt den Mauszeiger zu einer Sanduhr werden, um anzuzeigen, dass das Programm beschäftigt ist. Ansonsten passiert nichts, der Nutzer kann weiterhin Objekte anklicken und z.B. Texte eingeben.
MarkNotBusy nimmt ein MarkBusy zurück.

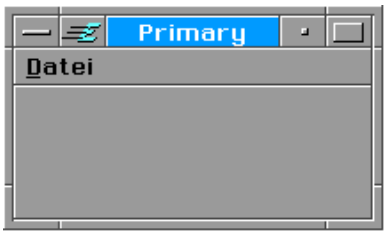
HoldUpInput, ResumeInput

HoldUpInput weist die UI an, Usereingaben nicht sofort an das Programm weiterzuleiten, sondern zwischenzuspeichern. Dabei wird der Mauszeiger **nicht** zur Sanduhr. Üblicher Weise wird deshalb HoldUpInput gemeinsam mit MarkBusy verwendet. ResumeInput weist die UI an, die zwischengespeicherten Usereingaben an das Programm weiterzuleiten, so dass sie behandelt werden können.
Verwenden Sie die beiden Methoden nur, wenn Sie keine andere Möglichkeit sehen, da der Nutzer schnell den Eindruck bekommen kann, das Ihr Programm "hängt".

IgnoreInput, AcceptInput

IgnoreInput weist die UI an, alle folgenden Eingaben zu blockieren. AcceptInput hebt diesen Zustand wieder auf. Alle zwischen IgnoreInput und AcceptInput erfolgten Eingaben (z.B. Klicks auf einen Button) sind **verloren**. Verwenden Sie diese Befehle nur als allerletzten Ausweg!

4.2 Primary



Objekte der Klasse **Primary** sind die "Hauptfenster" des Programms. Primaries haben links oben ein System-Menü, rechts oben die Schalter für "Minimieren" und "Maximieren" sowie bei Bedarf einen "Hilfe" Schalter (Fragezeichen).
Im Allgemeinen hat jedes BASIC Programm genau ein Primary-Objekt.

Focus und Target

Das Primary-Objekt ist ein Knoten in der Focus- und Target-Hierarchie. Es ist möglich zu überwachen, ob ein Primary-Objekt den Focus oder das Target hat, indem man einen Focus- bzw. Target-Handler schreibt. Die notwendigen Details zur Arbeit mit Focus und Target finden Sie im Kapitel 12 (Focus und Target) des Handbuchs "Spezielle Themen". Das Arbeiten mit Focus und Target ist etwas für erfahrene Programmierer und nur in wenigen Fällen notwendig. Eine Ausnahme bildet die Implementation von speziellen Menüs wie dem "Bearbeiten" Menü. Diesem Thema ist deswegen ein eigenes Kapitel ("Spezielle Themen", Kapitel 13) gewidmet.

Abstammung:

Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
BreakButton	BreakButton = TRUE	—
FileMenuChildren	FileMenuChildren = **<objektListe>**	—
Caption2$	—	lesen, schreiben
PrimaryFullScreen	PrimaryFullScreen	—
NoFileMenu	NoFileMenu	—
NoExpressMenu	NoExpressMenu	—
PrimaryNoHelpButton	PrimaryNoHelpButton	—

Spezielle Action-Handler: keine

BreakButton

Oftmals, besonders während der Fehlersuche, ist es erwünscht, einen laufenden Action-Handler abbrechen zu können, ohne gleich GEOS abzuwürgen. Das kann z.B. bei einer versehentlichen Endlosschleife der Fall sein, das Programm "hängt". Diese BREAK (Unterbrechung) genannte Funktion ist typisch für BASIC-Programme, bei "richtigen" GEOS-Programmen aber nicht vorhanden.

Das BreakButton-Statement fügt einen Break-Schalter zum Dateimenü des Primaries hinzu und aktiviert gleichzeitig die Tastenkombination Strg-B zum Auslösen eines BREAK.

Syntax UI-Code: **BreakButton** = TRUE

FileMenuChildren

Jedes Primary hat automatisch ein Datei-Menü. Dort ist per Default nur der "Beenden" Button enthalten. Häufig möchte man dort aber weitere Einträge vorsehen, z.B. zur Arbeit mit Dateien oder den Copyright-Dialog ("Über ... "). Das FileMenuChildren-Statement fügt die angegeben Objekte als Children in das Dateimenü ein.

Syntax UI-Code **FileMenuChildren** = **<ObjektListe>**

Beispiel:
Die in der Liste angegeben Objekt müssen natürlich extra vereinbart werden:

```
Primary MainPrimary
  Children = ...
  FileMenuChildren = AboutBox, SaveFileButton, LoadFileButton
END OBJECT
```

Die Anzahl der Objekte in einer einzigen FileMenuChildren-Liste ist auf 25 begrenzt. Wenn Sie mehr Children spezifizieren wollen können Sie, wie bei der Children-Anweisung, siehe Kapitel 2.1.2, mehrere FileMenuChildren-Anweisungen für ein Objekt verwenden.

Caption2$

Die Instancevariable Caption2$ ergänzt die Titelzeile des Primaryobjekts um einen weiteren Text. Üblicher Weise wird Caption2$ verwendet um den Namen des aktuellen Dokuments in der Titelzeile des Programms anzuzeigen. Caption2$ kann nicht im UI-Code verwendet werden.

Syntax Lesen: **<stringVar>** = **<obj>** . **Caption2$**
Schreiben: **<obj>**.**Caption2$** = **"text"**

Anpassen des Primary-Objekts

Das Primary-Objekt stammt von der GenericClass ab und erbt deswegen alle Fähigkeiten und Eigenschaften dieser Klasse. Das trifft insbesondere für die Geometrie-Fähigkeiten zu, wie z.B.

orientChildren
justifyChildren
MinimizeChildSpacing
DivideHeightEqually
DivideWidthEqually
childSpacing

Von besonderer Bedeutung für Primaries sind die Windows-Hints wie z.B.

SizeWindowAsDesired
NoTitleBar
NoSysMenu

Von der Klasse Display erbt das Primary die folgenden Instancevariablen. Eine ausführliche Beschreibung finden Sie im Kapitel 4.18.2 bei der Beschreibung der Display Klasse. Die Instancevariablen userDismissable, OnClose sowie die Methode Close werden nicht vererbt, das Primary Objekt implementiert hier sein eigenes Handling.

Variable	Syntax im UI-Code	Im BASIC-Code
minimizedState	minimizedState = TRUE | FALSE	lesen, schreiben
MinimizedOnStartup	MinimizedOnStartup	—
NotMinimizable	NotMinimizable	—
maximizedState	maximizedState = TRUE | FALSE	lesen, schreiben
MaximizedOnStartup	MaximizedOnStartup	—
NotMaximizable	NotMaximizable	—
NotResizable	NotResizable	—
NotRestorable	NotRestorable	—

Zusätzlich besitzen Primaries einige eigene Geometriefähigkeiten.

PrimaryFullScreen

Die Anweisung bewirkt, dass das Primary große Teile des Bildschirms einnimmt, so wie die "großen" Applikationen "GeoWrite" und "GeoDraw". Unten bleibt ein Rand für iconisierte Applikationen.

Syntax UI-Code:	**PrimaryFullScreen**

Beispiel:

```
Primary MyPrimary
  Children = ......
  PrimaryFullScreen
END Object
```

NoFileMenu

Die Anweisung verhindert, dass das Primary ein Datei-Menü hat. Es ist aber unwirksam, wenn es gleichzeitig mit **FileMenuChildren** oder **BreakButton** verwendet wird, da diese ein Dateimenü zwingend erfordern.

Syntax UI-Code: **NoFileMenu**

NoExpressMenu

Die Anweisung verhindert, dass sich das Express-Menü in der Titelzeile des Primaries ansiedelt.

Syntax UI-Code: **NoExpressMenu**

Beispiel:

```
Primary Primary2
  Children = ......
  NoExpressMenu
  NoFileMenu
END Object
```

PrimaryNoHelpButton

Syntax UI-Code: **PrimaryNoHelpButton**

Wenn Ihr Programm eine Hilfedatei hat, ist es sinnvoll dem Primary den HelpContext "TOC" (Table Of Contents = Inhaltsverzeichnis) zu geben, damit das Hilfesystem das Inhaltsverzeichnis findet.

```
Primary MyPrimary
  Children = ......
  helpContext$ ="TOC"
END Object
```

Diese Anweisung erzeugt jedoch gleichzeitig den Hilfebutton in der Titelzeile des Primaryobjekts (i.a ein blaues Fragezeichen). Wenn dies im Ausnahmefall stört können Sie es mit dem Hint **PrimaryNoHelpButton** unterdrücken.

Beachten Sie, dass der Name der Hilfedatei (Eintrag helpFile$) immer im Applicationobjekt erfolgen sollte, damit der vom Hilfesystem im Hilfefenster bereitgestellte Button "Inhalt" funktioniert.

```
Primary MyPrimary
  Children = ......
  helpContext$ ="TOC"
  PrimaryNoHelpButton
END Object
```

Besonderheiten des Primary Objekts

- Wenn Sie einem Primary keine Caption$ geben, so wird automatisch der Name des Programms genommen.

- Primary-Objekte sollten ständig im Objekttree eingebunden sein, Primaries, die kein Parent-Objekt haben, könnten die Systemstabilität beeinflussen. Verwenden Sie die Anweisung "myPrimary.visible = FALSE" bzw. im UI-Code "visible = FALSE", wenn sie ein Primary-Objekt verstecken wollen.

- Es ist möglich, wenn auch selten verwendet, dass ein Programm mehr als ein Primary-Objekt hat. Achtung: Sie sollten niemals mehrere **BreakButton** Statements verwenden, auch wenn Sie mehr als ein Primary haben.

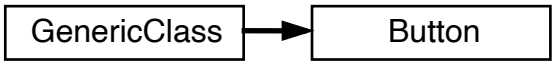
- Primaries sind Window-Objekte. Es gibt eine Menge Window-orientierte Hints und Methoden, die auf GenericClass Level definiert sind und mit Primaries zusammenarbeiten. Beispiele sind SizeWindowAsDesired, NoSysMenu, ExtendWindowToBottomRight, CenterWindow und ResizeWin.

4.3 Button

Ein Button ist eine Schaltfläche, die mit der Maus angeklickt oder mit der Tastatur aktiviert werden kann.

Buttons werden für Menü-Einträge oder als alleinstehende Schalter verwendet.

Abstammung:



Spezielle Instance-Variablen:

Variable	Syntax im UI-Code	Im BASIC-Code
ActionHandler	ActionHandler = **<Handler>** ActionHandler = BringUpHelp	nur schreiben —
actionData	actionData = **numWert**	lesen, schreiben
interactionCommand	interactionCommand = **numWert**	lesen, schreiben
BringsUpWindow	BringsUpWindow	—
IsDestructive	IsDestructive	—
unhandledEvents	—	nur lesen

Methoden:

Methode	Aufgabe
Activate	Auslösen des Buttons, als ob darauf geklickt wurde

Action-Handler-Typen:

Handler-Typ	Parameter
ButtonAction	(sender as object, actionData as integer)

ActionHandler

Die Instance-Variable **ActionHandler** enthält den Namen des aufzurufenden Actionhandlers. Dieser muss als **ButtonAction** vereinbart sein. Der Wert wird üblicherweise im UI-Code gesetzt.
Bei Bedarf kann er auch zur Laufzeit (im BASIC-Code) gesetzt, aber nicht gelesen werden.

Syntax	UI- Code:	**ActionHandler** = **<Handler>**
	Schreiben:	**<obj>**.**ActionHandler** = **<Handler>**

Beispiel: siehe unten

Syntax UI-Code: **ActionHandler** = **BringUpHelp**

Diese spezielle Syntax weist dem Button den von R-BASIC vordefinierten "Öffne die Hilfe" Handler zu. Wenn der Nutzer auf den Button klickt durchsucht der Handler den generic Tree nach einem Help Context und dem Namen einer Hilfedatei. Dazu durchsucht er zunächst den Button und dann der Reihe nach sein Parent, dessen Parent usw. Abschließend öffnet er die entsprechende Seite der Hilfe. Sehr oft hat daher der Button selbst einen Help Context gesetzt während die Hilfedatei vom Application Objekt bezogen wird. Dieses Verhalten kann man nutzen um ein Hilfemenü aufzubauen, dass spezielle Hilfeseiten direkt über Menüeinträge anzuspringen.

```
Button HelpButton
  Caption$="Hilfe zu irgend etwas"
  ActionHandler = BringUpHelp
  helpContext$ ="Help1"
  End Object
```

Natürlich kann man jedem Button auch eine eigene Hilfedatei zuordnen (helpFile$=..), wenn man das will.

actionData

Die Instance-Variable **actionData** enthält einen Integer-Wert, der bei Bedarf zur Identifizierung des Buttons oder sonstigen Zwecken herangezogen werden kann. Der Standard-Fall ist jedoch, dass der **actionData**-Wert nicht benutzt wird.

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

ButtonAction

Action-Handler für Buttons müssen als ButtonAction definiert werden.
Parameter: **sender**: Das Button-Objekt, das den Handler aktiviert hat
actionData: actionData-Wert des Buttons
Null, falls der Wert nicht gesetzt ist.

Beispiel 1: einfacher Button

UI-Code:

```
Button  MyButton
  Caption$ = "OK", 0
  ActionHandler = OKPressed
END Object
```

BASIC-Code:

```
ButtonAction   OKPressed
  < .. diverser Code ..>
  MsgBox "Operation erfolgreich."
END ACTION
```

Beispiel 2: Verwendung des actionData-Wertes

UI-Code:

```
Button  TestButton
  Caption$ = "Press mich"
  actionData = 255
  ActionHandler = Handler1
END OBJECT
```

BASIC-Code:

```
ButtonAction Handler1
DIM x, y as word
  x = actionData/2
  y = actionData + 12
  MsgBox "Gefundene Werte:"+ Str$(actionData)+ Str$(x) + Str$(y)
END ACTION
```

interactionCommand

Sehr häufig müssen Buttons in einer Dialogbox standardisierte Aktionen auslösen, wie z.B. das Schließen des Dialogs. Oder es ist nötig, dass eine Dialogbox einen Wert zurückgibt, z.B. ob der Nutzer auf "Ja", "Nein" oder "Abbrechen" geklickt hat. Für diesen Zweck kann man einem Button einen "Interactions-Kommando" - Wert (einen WORD-Wert) zuordnen, so dass das GEOS-System automatisch weiß, was es zu tun hat.

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

Da die Verwendung von **interactionCommand**-Werten nur im Zusammenhang mit Dialogboxen sinnvoll ist, werden sie dort ausführlich besprochen. Zur Vereinfachung finden Sie unten trotzdem eine Tabelle der verfügbaren Interaction-Command-Werte. Beachten Sie, dass ein Button nur **entweder** einen action-Handler **oder** ein interactionCommand haben kann. Die Zuweisung des Einen im BASIC-Code löscht jeweils das Andere.

interactionCommand	Wert	Bedeutung
IC_CLOSE	1	Dialog schließen
IC_APPLY	3	Änderungen anwenden
IC_RESET	4	Dialog zurücksetzen
IC_OK	5	Verwendet für "OK"
IC_YES	6	Verwendet für "Ja"
IC_NO	7	Verwendet für "Nein"
IC_STOP	8	Verwendet für "Stop" oder "Abbrechen"
IC_HELP	10	Button ersetzt den "Hilfe" Button

BringsUpWindow

Dieser Hint platziert eine Ellipse "..." hinter dem Button-Text um anzuzeigen, dass der Button eine Dialogbox oder ähnliches auf den Schirm bringt. Damit sieht ein Menüeintrag genau so aus, als sei die Dialogbox das direkte Child des Menüs.

Syntax UI-Code: **BringsUpWindow**

IsDestructive

Ähnlich wie der Hint CannotBeDefault für Groups soll IsDestructive verhindern, dass der Nutzer den Button als "Default-Aktion" per Enter- oder Leertaste aktiviert. Verwenden Sie diesen Hint, wenn der Button eine potentiell gefährliche (destruktive) Aktion auslöst.

Syntax UI-Code: **IsDestructive**

unhandledEvents

Enthält, wie oft der Button "aktiviert" wurde (z.B. durch Anklicken mit der Maus), ohne dass der zugehörige ActionHandler aufgerufen werden konnte. Dass passiert i.A. wenn noch ein anderer ActionHandler läuft, während der Button aktiviert wurde. Da alle ActionHandler nacheinander (im gleichen Thread) abgearbeitet werden sie in der Reihenfolge abgearbeitet in der sie auftreten, ohne sich gegenseitig zu unterbrechen.
Achtung! Die Instancevariable **unhandledEvents** enthält immer den Wert Null, wenn dem Button kein **ActionHandler** zugewiesen wurde.
Sie können diese Instancevariable benutzen, wenn Sie einen lang andauernden Prozess vorzeitig abbrechen wollen, aber dafür keinen Progress-Dialog (siehe Kapitel 4.6.6.4) einsetzen möchten. Bedenken Sie aber, dass die entsprechenden Ereignisse bereits in der Ereigniswarteschlage (Event queue) abgelegt sind und daher auf jeden Fall später noch abgearbeitet werden.

Syntax BASIC-Code:	**<numVar>** =**<obj>**.**unhandledEvents**

Activate

Diese Methode bewirkt, dass der Button aktiviert wird, so als ob der User direkt darauf geklickt hat. Der Action-Handler oder das InteractionCommand des Buttons wird ausgeführt.

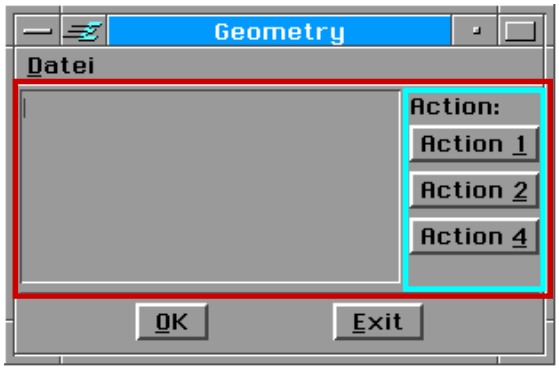
Syntax BASIC-Code:	**<obj>**.**Activate**

Beispiel:

```
IF x > 0 THEN MyButton.Activate
```

4.4 Group

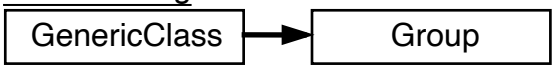
Groups (Gruppen) haben einen Hauptzweck: Sie dienen dazu andere Objekte anzuordnen. Durch die geschickte Verwendung von Groups geben Sie als Programmierer ihrem Programm das Aussehen, das Sie wünschen.



Diese Anwendung, die Sie vielleicht aus dem Kapitel über das Geometriemanagement kennen, besteht aus einer Group (rot, oben) und einer Reply-Bar (unten, ebenfalls ein Group-Objekt), die vertikal (untereinander) angeordnet sind. Die rot markierte Gruppe besteht aus einem Text-Objekt und einer weiteren Group, hier blau markiert.

Beide Objekte sind horizontal angeordnet. Die rechte (blaue) Group enthält letztlich die Action-Buttons, die untereinander angeordnet sind.

Abstammung



Spezielle Instance-Variablen:

Hint	Syntax im UI-Code	Im BASIC-Code
CannotBeDefault	CannotBeDefault	—

CannotBeDefault

Sehr häufig, z.B. in Dialog-Boxen, gibt es einen Button oder ein anderes Objekt, dass über den Standard-Weg "Leertaste" oder "Entertaste" aktiviert wird. Viele Nutzer neigen dazu, beim Erscheinen einer Dialogbox erst einmal auf "Enter" zu drücken und damit diesen "Default"-Button zu aktivieren. Der Hint CannotBeDefault verhindert, dass die Children der Group über diesen Weg "per Default" aktiviert werden können. Sie können Groups mit diesem Hint versehen, wenn die darin enthaltenen Objekte (z.B. Buttons) potentiell gefährliche Aktionen auslösen können.

Syntax UI-Code: **CannotBeDefault**

Groups stammen von der GenericClass ab und erben damit alle Eigenschaften und Fähigkeiten dieser Klasse. Besonders interessant sind in diesem Zusammenhang die Fähigkeiten zum Geometriemanagement, die im Kapitel 3.3 ausführlich besprochen und von Group-Objekten sehr häufig verwendet werden. Der Einfachheit halber sind die aus der Sicht einer Group wichtigsten - aber nicht alle - Hints zum Geometriemanagement hier noch einmal aufgeführt:

Anordnung der Children

Hint	Syntax im UI-Code	Im BASIC-Code
orientChildren	orientChildren = **numWert**	lesen, schreiben
justifyChildren	justifyChildren = **numWert**	lesen, schreiben
childSpacing	childSpacing = **numWert**	lesen, schreiben
MinimzeChildSpacing	MinimzeChildSpacing	——
IncludeEndsInChildSpacing	IncludeEndsInChildSpacing	——
wrapAfterChild	wrapAfterChild = **numWert**	lesen, schreiben

Objektgröße

Hint	Syntax im UI-Code	Im BASIC-Code
DivideHeightEqually	DivideHeightEqually	——
DivideWidthEqually	DivideWidthEqually	——
ExpandWidth	ExpandWidth	—
ExpandHeight	ExpandHeight	—
initialSize	initialSize = **x**, **y** [, **count**]	lesen, schreiben
minimumSize	minimumSize = **x**, **y** [, **count**]	lesen, schreiben
maximumSize	maximumSize = **x**, **y** [, **count**]	lesen, schreiben
fixedSize	fixedSize = **x**, **y** [, **count**]	lesen, schreiben

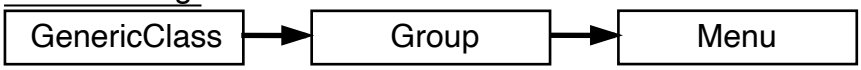
Spezielle Attribute

Hint	Syntax im UI-Code	Im BASIC-Code
DrawInBox	DrawInBox	—
MakeToolbox	MakeToolbox	—
MakeReplyBar	MakeReplyBar	—
NoSeparatorLine	NoSeparatorLine	—

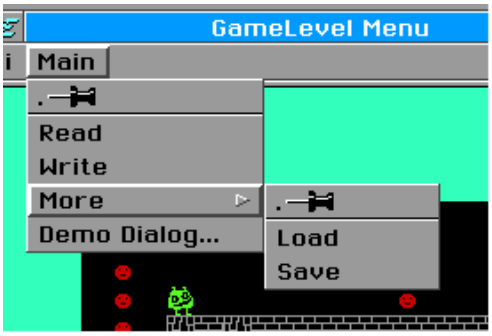
4.5 Menu

Menüs sind der übliche Weg, auf dem der Nutzer unterschiedliche Programmfunktionen anwählen kann.

Abstammung:



Spezielle Instance-Variablen: keine



Die Menü-Einträge sind die Children eines Menüs. In vielen Fällen sind dies Buttons. Verschachtelte Menüs (Sub-Menüs, siehe Bild) erhalten Sie, wenn sie andere Menüs als Menü-Einträge verwenden. Verwenden Sie Dialoge als Menü-Einträge erzeugt das System selbständig einen Button im Menü, der den Dialog öffnet. Sie dürfen jedoch auch beliebige andere Objekte, etwa Groups oder Listen-Objekte als Children von Menüs verwenden.

Groups erzeugen automatisch einen Trennstrich, um sich vom Rest des Menüs abzuheben. Wünschen Sie diesen nicht, so verwenden Sie den Hint **NoSeparatorLine** für die Group.

Beispiel: UI-Code-Fragment für das im Bild gezeigte Menü. Natürlich müssen alle Buttons einen Action-Handler haben, auch wenn sie hier nicht aufgeführt sind. Die Trennung von UI-Anweisungen mit einem Doppelpunkt ist, genau wie im BASIC-Code, erlaubt.

```
Menu MainMenu
  Caption$ = "Main"
  Children = ReadButton, WriteButton, SubMenu, DemoDialog
  END Object

Menu Submenu
  Caption$ = "More"
  Children = LoadButton, SaveButton
  END Object
Button ReadButton
  Caption$ = "Read"
  ActionHandler = ReadHandler
  END Object
Button WriteButton : Caption$ = "Write" : END Object
Button LoadButton : Caption$ = "Load" : END Object
Button SaveButton : Caption$ = "Save" : END Object

Dialog DemoDialog
  Caption$ = "Demo Dialog"
  dialogType = DT_NOTIFICATION     ' siehe nächstes Kapitel
  END Object
```

(Leerseite)