Commodity

COLLABORATORS			
	TITLE : Commodity		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 23, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Commodity

1.1 Commodity V1.00

```
Pure Basic - Commodity V1.00

   Die Commodities sind ein Weg, unter dem AmigaOS Programme zu
   verwalten. Ein Programm, das ein Commodity werden möchte, muß
   einige Regeln beachten, und es kann später über ein nützliches
   Programm, genannt 'Exchange', kontrolliert werden. Sie können
   damit das Programm aktivieren/deaktivieren, es verstecken oder
   anzeigen, ein Break (Abbruch) Signal senden, es beenden und
   vieles mehr. Diese PureBasic Library bietet Ihnen die Möglichkeit,
   die Commodity Features einfach Ihrem Programm hinzuzufügen.

Befehlsübersicht:

   ActivateCommodity
   ActivateCommodityObject
   ActivateCommodityTranslater
   AddCommodityInputEvent
   ChangeCommodityFilter
   ChangeCommodityFilterIX
   ChangeCommodityTranslater
   CommodityCtrlCSignal
   CommodityEvent
   CommodityID
   CommoditySignal
   CommodityType
   CreateCommodityObject
   FreeCommodityObject
   InitCommodity
   WaitCommodityEvent


   Commodity Demo 1
   Commodity Demo 2
```

1.2 activatecommodity

```
  SYNTAX
ActivateCommodity(Status.l)

  STATEMENT
Aktiviert oder deaktiviert das Commodity, welches alle erstellten
Objekte beinhaltet.

Wenn das Commodity deaktiviert ist, empfängt es nur CxMessages
(Nachrichten) vom Befehlstyp (Command type) vom Commodity 'Exchange';
CxMessages vom Ereignistyp (Event type) werden nicht verarbeitet.

Status
Aktiviert das Commodity durch Setzen des Status auf TRUE bzw.
deaktiviert es mit dem Status FALSE.
```

1.3 activatecommodityobject

```
  SYNTAX
ActivateCommodityObject(#Obj.l,Status.l)

  STATEMENT
Deaktiviert oder aktiviert ein Objekt, welches mit CreateCommodityObject()
erstellt wurde.

Ein deaktiviertes Objekt "schläft", d.h. es verarbeitet keine CxMessages,
bis es mit einem ActivateCommodityObject(#Obj, TRUE) "aufgeweckt" wird.

Dieses Statement hat keine Wirkung, wenn das Objekt unbenutzt ist.

#Obj
zu aktivierendes oder zu deaktivierendes Objekt

Status
Deaktiviert das Objekt durch Setzen von Status auf FALSE bzw. aktiviert
das Objekt durch Setzen von Status TRUE.
```

1.4 activatecommoditytranslater

```
  SYNTAX
ActivateCommodityTranslater(#Obj.l,Status.l)

  STATEMENT
Deaktiviert oder Aktiviert den Translater (Übersetzer) in einem Objekt.

Ein aktivierter Translater verändert die Eingabenachrichten (Input events)
auf zwei Wegen, sie können beseitigt werden oder durch eine neue Eingabe-
Nachricht ersetzt werden. Wenn keine dieser Funktionen nützlich ist, dann
deaktivieren Sie einfach den Translater.

Dieses Statement hat keine Wirkung, wenn das Objekt unbenutzt ist.
```

```
#Obj
zu benutzendes Objekt

Status
Deaktiviert den Translater durch Setzen des Status auf FALSE und aktiviert
den Translater durch Setzen von Status TRUE.
```

1.5 addcommodityinputevent

```
  SYNTAX
AddCommodityInputEvent(*InputEvent)

  STATEMENT
Fügt an den Eingabestrom der Nachrichten eine einzelne oder eine ganze
Kette von Nachrichten (Input events) an.

*InputEvent
Dies ist ein Zeiger auf eine InputEvent Struktur oder eine Kette von
InputEvent Strukturen. Nach dem Funktionsaufruf ist dieser zur freien
Benutzung bestimmt.
```

1.6 changecommodityfilter

```
  SYNTAX
Result.b = ChangeCommodityFilter(#Obj.l,Filter$)

  FUNCTION
Verändert die Filter Optionen für ein Objekt.

Diese Funktion hat keine Wirkung, wenn das Objekt unbenutzt ist.

#Obj
Das Objekt, dessen Filter verändert werden soll.

Filter$
Der String, welcher die neuen Filter Optionen beschreibt.

Result
Ergibt dieses TRUE, ist der String mit der Filter Beschreibung fehlerhaft
und das Objekt verarbeitet bis zum nächsten erfolgreichen Aufruf dieser
Funktion oder von ChangeCommodityFilterIX() keine Nachrichten (CxMessages)
mehr.
```

1.7 changecommodityfilterix

```
  SYNTAX
Result.b = ChangeCommodityFilterIX(Obj.l,*InputXpression)

  FUNCTION
Verändert die Filter Optionen mit einer InputXpression Struktur.
```

```
Diese Funktion hat keine Wirkung, wenn das Objekt unbenutzt ist.

#Obj
Das Objekt, dessen Filter verändert werden soll.

*InputXpression
Ein Zeiger auf eine InputXpression Struktur, welche die neuen Filter
Optionen beschreibt. Die Struktur kann nach dem Funktionsaufruf beliebig
weitergenutzt werden.

Result
Ist dieses True, dann war in der InputXpression Struktur etwas enthalten,
das keinen Sinn machte. Das Objekt verarbeitet bis zum nächsten erfolg-
reichen Aufruf dieser Funktion oder von ChangeCommodityFilter() keine
Nachrichten (CxMessages) mehr.
```

1.8 changecommoditytranslater

```
  SYNTAX
ChangeCommodityTranslater(#Obj.l,*InputEvent)

  STATEMENT
Verändert den InputEvent vom Translater, welcher jeden CxMessage Input
Event (Nachricht) ersetzt.

Diese Funktion hat keine Wirkung, wenn das Objekt unbenutzt ist.

#Obj
zu benutzendes Objekt

*InputEvent
Dies ist ein Zeiger auf eine InputEvent Struktur oder eine Kette von
InputEvent Strukturen. Nach dem Funktionsaufruf ist dieser zur freien
Benutzung bestimmt.
```

1.9 commodityctrlcsignal

```
  SYNTAX
Result.w = CommodityCtrlCSignal()

  FUNCTION
Wenn eine Commodity Ereignis (Event) passierte, sieht diese Funktion nach,
ob die CTRL C Tasten gedrückt wurden.

Result
Ergibt TRUE, wenn CTRL C gedrückt wurde, andernfalls FALSE.
```

1.10 commodityevent

```
  SYNTAX
Result.w = CommodityEvent()

  FUNCTION
Die Funktion überprüft, ob irgendein Commodity Ereignis (Event) auftrat.

Ein Commodity Ereignis kann eines der folgenden sein: ein aktives Objekt
erhält eine gesuchte CxMessage (Nachricht); der Anwender drückt einen
Schalter im Commidity Exchange; oder der Anwender drückt in einer
CLI Umgebung die Tastenkombination CTRL C.

CommodityEvent() wartet nicht auf passierende Ereignisse, wie
WaitCommodityEvent() - dies ist nützlich, wenn die Ereignisschleife
(Eventloop) weiterlaufen soll.

Result
Ergibt TRUE bei irgeneinem Commodity Ereignis, andernfalls FALSE.

EventLoop
```

1.11 commodityid

```
  SYNTAX
Result.w = CommodityID()

  FUNCTION
Diese Funktion gibt die ID Nummer des Objekts zurück, das eine CxMessage
(Nachricht) oder einen Befehl vom Commodity Exchange erhalten hat.

Result
Dies ist dasselbe wie #param1 bei CreateCommodityObject() wenn das Objekt
erstellt wurde, aber es kann auch ein Befehl vom Commodity Exchange sein,
wenn das Ergebnis von CommodityType() vom Befehlstyp ist.
```

1.12 commoditysignal

```
  SYNTAX
Result.w = CommoditySignal()

  FUNCTION
Wenn ein Commodity Ereignis auftrat, überprüft diese Funktion, ob ein
Signal von einem Objekt oder vom Commodity Exchange kam.

Result
Dieses ist TRUE, wenn ein Signal von einem Objekt beim Commodity eintraf
oder das Commodity Exchange einen Befehl sandte, andernfalls ist es FALSE.
```

1.13 commoditytype

```
  SYNTAX
Result.w = CommodityType()

  FUNCTION
Diese Funktion gibt den Nachrichtentyp einer CxMessage zurück.

Die CxMessage (Nachricht) ist entweder vom Befehls-Typ oder vom
Nachrichten-Typ. Ein Befehlstyp tritt auf, wenn der Anwender einen
Schalter im Commodity Exchange gedrückt hat und ein Nachrichtentyp,
wenn ein Objekt eine CxMessage erhält.

Result
Ergibt den Nachrichtentyp.
```

1.14 createcommodityobject

```
  SYNTAX
Result.b = CreateCommodityObject(#Obj.l,Filter$,*InputEvent)

  FUNCTION
Diese Funktion erstellt ein Objekt. Das Objekt wird im aktiven (enabled)
Status erstellt und fängt sofort nach Aktivierung des Commodity mit
dem Verarbeiten von CxMessages (Nachrichten) an.

Wird das Objekt bereits benutzt, hat die Funktion keinen Einfluß darauf
und erstellt einfach ein neues Objekt ohne das alte zu löschen. Danach
gibt es keine Möglichkeit mehr, das alte Objekt zu löschen oder zu
verändern.

Ein Objekt besteht aus drei Teilen.

* Dem Filter, dessen einziger Zweck es ist, die Art der Nachrichten
  (CxMessages), an denen das Objekt interessiert ist, herauszufiltern.
  Der Filter während des Programmablaufs verändert werden.

* Dem Sender, dessen einziger Zweck es ist, dem Commodity zu signalisieren,
  wenn es eine CxMessage erhält.

* Dem Übersetzer, dessen einziger Zweck es ist, alle CxMessage Eingabe-
  Ereignisse (Input events) in neue zu übersetzen - der Übersetzer
  (Translater) muß hierfür aktiviert sein. Der Übersetzer kann während
  des Programmablaufs verändert werden.

#Obj
Dies ist die benötigte Objekt Nummer und sollte nicht höher sein als
#param1 beim Aufruf von InitCommodity().

Filter$
Dieser String legt die Filter Optionen fest, eine Beschreibung über was
dieses Objekt informiert werden will.

*InputEvent
Dies ist ein Zeiger auf eine InputEvent Struktur oder einer Kette von
InputEvent Strukturen, das tatsächliche Ereignis (InputEvent) wird
```

```
gelöscht und durch dieses neue ersetzt.

Ist der Zeiger gleich NULL, wird das tatsächliche Ereignis einfach
gelöscht und kein anderes Commodity oder das OS erfährt davon.

Result
Ergibt dieses #COERR_ISNULL (1), konnte das Objekt nicht erstellt werden,
ist das Ergebnis dagegen gleich #COERR_BADFILTER (4), wurde das Objekt
erstellt, kann aber bis zu einer Veränderung des Filters mittels
ChangeCommodityFilter() oder ChangeCommodityFilterIX() keine CxMessages
verarbeiten.
```

1.15 freecommodityobject

```
  SYNTAX
FreeCommodityObject(#Obj.l)

  STATEMENT
Gibt ein deaktiviertes oder aktiviertes Objekt frei.

Dieses Statement hat keine Wirkung, wenn das Objekt unbenutzt ist.

#Obj
Das freizugebende Objekt.
```

1.16 initcommodity

```
  SYNTAX
Result.b = InitCommodity(Objects.l,Name$,Title$,Description$,
                         Flag.w,Priority.b)

  FUNCTION
Diese Funktion erstellt die Grundlage für ein Commodity.

Das Commodity wird in einem deaktivierten Zustand erstellt. Wenn Sie also
einige Objekte erstellt haben, aktivieren Sie sie mit dem Befehl
ActivateCommodity(TRUE).

Dies ist die Initialisierungs-Routine und sollte immer zuerst aufgerufen
werden. Sie kann zur Zeit nur einmal aufgerufen werden. Wenn also beim
Aufruf dieser Funktion ein Fehler auftritt, dann sollte das Programm
stets beendet werden.

Objects
Die Zahl der benötigten Objekte. Maximale Anzahl ist 2046.

Name$
Dieser String beschreibt den Namen des Commodity, welcher für jedes Commodity
einmalig sein sollte.

Title$
Dieser String beschreibt den Titel, welcher während des Programmablaufs im
```

```
Fenster des Commodity 'Exchange' angezeigt wird

Description$
Dieser String definiert die Beschreibung des Commodity, welche während des
Programmablaufs im Fenster des Commodity 'Exchange' angezeigt wird.

Flag
Wird dieser auf #COF_SHOW_HIDE gesetzt, dann soll das Commodity das GUI
(Oberfläche) zeigen/verstecken (show/hide), wenn der Anwender in Exchange
die Schalter "Anzeige sichtbar" bzw. "Anzeige verbergen" drückt. Außerdem
wird die Anzeige sichtbar gemacht, wenn das Commodity mehr als einmal
gestartet wird, anstelle es wie ein Commodity ohne GUI zu beenden.

Um alles richtig zu machen, sollte das Commodity das Tooltype CX_POPUP
einlesen und nachsehen, ob der Anwender beim ersten Start des Commodity
die Anzeige des GUI wünscht.

Priority
Das Commodity wird in die Liste der Commodities eingefügt und der Platz
ist dabei von der Priorität - im Bereich von -128 bis 127 - abhängig.
Eine höhere Priorität verschafft dem Commodity einen höheren Platz in
der Commodities Liste, welches dadurch die CxMessages früher erhält.

Um alles richtig zu machen, sollte das Commodity das Tooltype CX_PRIORITY
einlesen und die vom Anwender definierte Priorität benutzen.

Result
Wenn das Commodity nicht erstellt werden konnte, ist dieses FALSE und als
einziger Weg bleibt dann nur noch das Beenden des Programms.
```

1.17 waitcommodityevent

```
  SYNTAX
WaitCommodityEvent()

  STATEMENT
Diese Funktion überprüft, ob irgendein Commodity Ereignis (Event) auftrat.

Ein Commodity Ereignis ist eines der folgenden: wenn das aktive Objekt
eine gesuchte CxMessage erhält; wenn der Anwender im Commodity Exchange
einen Schalter betätigt; oder wenn der Anwender in einer CLI Umgebung
CTRL C drückt.

WaitCommodityEvent() wartet auf passierende Ereignisse, nicht wie
CommodityEvent(), - dies ist nützlich, um Rechenzeit zu sparen.

EventLoop
```

1.18 filterstrings

```
  [Class]  {[-] (Qualifier|Synonym)}  [[-] upstroke]  [highmap|ANSICode]
```

```
       Class
  Qualifier|Synonym
      upstroke
  highmap|ANSICode
```

```
Einige einfache Eingabe Beschreibungsstrings:
---------------------------------------------
"rawkey upstroke a"

"rawkey -upsroke f1"

"timer"

"diskremoved"

"rawkey leftbutton f2"
```

1.19 class

Class (Klasse) kann einer der Klassenstrings aus der nachfolgenden Tabelle sein.

Class String
rawkey
timer
diskremoved
diskinserted

1.20 qualifier|synonym

Qualifier ist einer der Qualifier Strings aus der nachfolgenden Tabelle.

A dash preceding the qualifier string tells the filter object not to care if that qualifier is present in the input event. Notice that there can be more than one qualifier (or none at all) in the input description string.

Qualifier String
lshift
rshift
capslock
control
lalt
ralt
lcommand
rcommand
numericpad
repeat

```
   midbutton
   rbutton
   leftbutton
   relativemouse

Synonym is one of the synonym strings from the table below.  These
strings act as synonyms for groups of qualifiers. A dash preceding
the synonym string tells the filter object not to care if that
synonym is present in the input event.  Notice that there can be more
than one synonym (or none at all) in the input description string.

   Synonym String
   --------------
   shift        look for either shift key
   caps         look for either shift key or capslock
   alt          look for either alt key
```

1.21 upstroke

```
Upstroke is the literal string "upstroke".  If it is present alone the
filter considers only upstrokes, if it's absent the filter considers only
downstrokes and if preceded by a dash the filter considers both upstrokes
and downstrokes.
```

1.22 highmap|ansicode

```
Highmap is one of the following strings:

   space , backspace , tab , enter , return , esc , del , help,
   up , down , right , left,
   f1 , f2 , f3 , f4 , f5 , f6 , f7 , f8 , f9 , f10.

For some reason commodities.library accept f11 and f12 as
valid keys.

ANSICode is a single character for example 'a' .
```

1.23 eventloop1

```
 Repeat

   VWait()  ; - to slow down the loop.

   If CommodityEvent()

     If CommoditySignal()
       Other code...
       ... ...
     EndIf
```

```
    If CommodityCtrlCSignal()
      quit=1
    EndIf

  EndIf

Until quit = 1
```

1.24 eventloop2

```
Repeat

  WaitCommodityEvent()

  If CommoditySignal()
    Other code...
    ... ...
  EndIf

  If CommodityCtrlCSignal()
    quit=1
  EndIf

Until quit = 1
```