

Commodity

COLLABORATORS

	<i>TITLE :</i> Commodity		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		January 19, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Commodity	1
1.1	Commodity V1.00	1
1.2	activatecommodity	1
1.3	activatecommodityobject	2
1.4	activatecommoditytranslator	2
1.5	addcommodityinputevent	2
1.6	changecommodityfilter	3
1.7	changecommodityfilterix	3
1.8	changecommoditytranslator	3
1.9	commodityctrlsignal	4
1.10	commodityevent	4
1.11	commodityid	5
1.12	commoditysignal	5
1.13	commoditytype	5
1.14	createcommodityobject	5
1.15	freecommodityobject	6
1.16	initcommodity	7
1.17	waitcommodityevent	8
1.18	filterstrings	8
1.19	class	8
1.20	qualifierlsynonym	9
1.21	upstroke	9
1.22	highmaplansicode	9
1.23	eventloop1	10
1.24	eventloop2	10

Chapter 1

Commodity

1.1 Commodity V1.00

PureBasic - Commodity V1.00

'Commodity' est un terme définissant une manière de contruire une application sous l'AmigaOS. Une commodité respecte un certain nombre de règles établies, qui permettent par la suite de piloter le programme à distance grâce au programme 'Exchange'. Par exemple, vous pouvez afficher/cacher la fenêtre, suspendre l'exécution ou fermer l'application. Une commodité peut en plus gérer facilement les signaux (CTRL C, ...) et les HotKeys (ALT F7, ...).

Commandes disponibles:

ActivateCommodity
ActivateCommodityObject
ActivateCommodityTranslator
AddCommodityInputEvent
ChangeCommodityFilter
ChangeCommodityFilterIX
ChangeCommodityTranslator
CommodityCtrlCSignal
CommodityEvent
CommodityID
CommoditySignal
CommodityType
CreateCommodityObject
FreeCommodityObject
InitCommodity
WaitCommodityEvent

Commodity Demo 1
Commodity Demo 2

1.2 activatecommodity

Syntaxe

```
ActivateCommodity (Status.1)
```

STATEMENT

Active ou désactive la commodité, y compris tous les objets préalablement créés. Quand la commodité est désactivée, alors les messages recus seront limités au type 'Command'. Les messages de type 'Event' sont ignorés.

Status: Si status = 1, la commodité sera activé, sinon elle sera désactivée.

1.3 activatecommodityobject

Syntaxe

```
ActivateCommodityObject (#Obj.1, Status.1)
```

STATEMENT

Active ou désactive un objet créé par CreateCommodityObject() .

Un objet désactivé est en quelque sorte endormi, il ne gère plus les messages de la commodité jusqu'à ce qu'il soit réactivé.

#Obj: Identifiant numérique de l'objet

Status: TRUE: Active l'objet, FALSE: Désactive l'objet.

1.4 activatecommoditytranslator

Syntaxe

```
ActivateCommodityTranslator (#Obj.1, Status.1)
```

STATEMENT

Active ou désactive le 'Translator' pour un objet donné.

Si le 'Translator' est activé, alors le CxMessage peut être changé de 2 façons: soit il est éliminé, soit il est remplacé par un nouvel 'Event'. Quand aucun des deux n'est utile, désactivez le 'Translator'.

#Obj: Identifiant numérique de l'objet

Status: Active le 'Translator' si status=TRUE; désactive le 'Translator' si status=FALSE.

1.5 addcommodityinputevent

Syntaxe

```
AddCommodityInputEvent (*InputEvent)
```

STATEMENT

Ajoute un nouvel 'InputEvent' ou une chaîne d'InputEvent à la liste actuelle.

*InputEvent: Pointeur vers une structure 'InputEvent'. Vous pouvez réutiliser librement cette structure après l'appel de la fonction.

1.6 changecommodityfilter

Syntaxe

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

Résumé:

Change le filtre de l'objet donné. Si le résultat est TRUE, alors la nouvelle description du filtre est fausse.

#Obj: Identifiant numérique de l'objet.

Filter\$: chaîne de caractères décrivant le nouveau filtre.

1.7 changecommodityfilterix

Syntaxe

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

Résumé

Change the Filter conditions with an InputXpression Structure.

This Résumé doesn't care if the Object is unused.

#Obj

The Object to change the Filter for.

*InputXpression

A pointer to a InputXpression Structure that describes the new Filter conditions, the Structure is free to use again after this call.

Result

If it's TRUE there was something that didn't make sense in the InputXpression and the Object won't process any CxMessage until either: this Résumé, or ChangeCommodityFilter() succeeds the next time.

1.8 changecommoditytranslator

Syntaxe

```
ChangeCommodityTranslator(#Obj.l,*InputEvent)
```

STATEMENT

Change the Translator's InputEvent which replaces every CxMessage input event received.

This statement doesn't care if the Object is unused.

#Obj

Object to use.

*InputEvent

This is a pointer to an InputEvent Structure or a chain of InputEvent Structures and it is free to use again after this call.

1.9 commodityctrlcsignal

Syntaxe

```
Result.w = CommodityCtrlCSignal()
```

Résumé

When a Commodity event has occurred this Résumé checks to see if the Ctrl C keys were pressed.

Result

This is TRUE if Ctrl C was pressed else it's FALSE.

1.10 commodityevent

Syntaxe

```
Result.w = CommodityEvent()
```

Résumé

This Résumé checks if any Commodity event has occurred.

A Commodity event can be one of the following: if any enabled Object receives the CxMessage it's looking for; if the user presses a button in Commodities Exchange; or, if the user presses Ctrl C in a CLI environment.

CommodityEvent() doesn't wait for events to happen, unlike WaitCommodityEvent() - this is useful when the eventloop should go on.

Result

This is TRUE for any Commodity event else it's FALSE.

EventLoop

1.11 commodityid

Syntaxe

```
Result.w = CommodityID()
```

Résumé

This Résumé return the ID of the Object that received a CxMessage or a command from Commodities Exchange.

Result

This is the same as #param1 in CreateCommodityObject() when the Object is created, but it could also be a command from Commodities Exchange if the result from CommodityType() is of Command Type.

1.12 commoditysignal

Syntaxe

```
Result.w = CommoditySignal()
```

Résumé

When a Commodity event has occurred this Résumé checks if the signal came from an Object or from Commodities Exchange.

Result

This is TRUE if an Object signaled the Commodity or if Commodities Exchange send a command, else it's FALSE.

1.13 commoditytype

Syntaxe

```
Result.w = CommodityType()
```

Résumé

This Résumé return the Message Type of a CxMessage.

The CxMessage is either of Command Type or Event Type, the Command Type comes when the user presses a button in the Commodities Exchange and the Event Type when an Object receives a CxMessage.

Result

This is the Message Type.

1.14 createcommodityobject

Syntaxe

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

Résumé

This Résumé creates an Object. The Object is created in enabled state

and starts to process CxMessages immediately if the Commodity is enabled.

If the Object is already in use the Résumé doesn't care and just creates a new Object without deleting the old one, after that there is no way to delete or change the old Object.

An Object consists of three parts.

- * The Filter, whose only purpose is to filter out the kind of CxMessage the Object is interested in. The Filter can be changed at runtime.
- * The Sender, whose only purpose is to signal the Commodity when it receives a CxMessage.
- * The Translator, whose only purpose is to translate every CxMessage input event this Object receives, into a new one - the Translator needs to be enabled to do this. The Translator can be changed at runtime.

#Obj

This is the Object number required and should not be higher then #param1 in InitCommodity().

Filter\$

This string sets the Filter conditions, a description of what this Object wants to know about.

*InputEvent

This is a pointer to an InputEvent Structure or a chain of InputEvent Structures, the real input event is deleted and replaced by this new one.

If the pointer is zero the real input event is just deleted, no other Commodity or the OS will know about it.

Result

If this is #COERR_ISNULL (1) then the Object could not be created but if it's #COERR_BADFILTER (4) the Object is created but it doesn't process any CxMessages until the Filter is changed with ChangeCommodityFilter() or ChangeCommodityFilterIX().

1.15 freecommodityobject

Syntaxe

FreeCommodityObject(#Obj.1)

STATEMENT

Free a Disabled or Enabled Object.

This statement doesn't care if the Object is unused.

#Obj

The Object to free.

1.16 initcommodity

Syntaxe

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

Résumé

This Résumé creates the basic stuff for a Commodity.

The Commodity is created in a disabled state so after you've created some Objects, enable it with `ActivateCommodity(TRUE)`.

This is the Initroutine and should always be called first and can only be called once, at the moment, so if there is a failure with this call, then the program should always quit.

Objects

The number of Objects required, Max Objects is 2046.

Name\$

This string describes the name of the Commodity and should be unique for each Commodity.

Title\$

This string describes the title that shows up in the window of Commodities Exchange when the Commodity is running.

Description\$

This string describes the description of the Commodity that shows up in the window of Commodities Exchange when the Commodity is running.

Flag

If this is set to `#COF_SHOW_HIDE` then the Commodity should show/hide a GUI when the user presses show interface/hide interface buttons in Commodities Exchange and even let the GUI pop up when the Commodity is started more than once, instead of letting it quit as a none GUI Commodity should do.

To do it properly the Commodity should read ToolType `CX_POPUP` and see if the user wants the GUI to pop up when the Commodity is started for the first time.

Priority

The Commodity is inserted in the `commoditys` list and the place depends on the priority - ranging between -128 and 127. - A higher priority gives the Commodity a earlier place in `commoditys` list and by that it gets the `CXMessages` earlier.

To do it properly the Commodity should read the ToolType `CX_PRIORITY` and use the priority specified by the user.

Result

If the Commodity could not be created this is `FALSE` and the only thing is to quit.

1.17 waitcommodityevent

Syntaxe

```
WaitCommodityEvent()
```

STATEMENT

This Résumé checks if any Commodity event has occurred.

A Commodity event is one of the following: if an enabled Object receives the CxMessage it's looking for; if the user presses a button in Commodities Exchange; or, if the user presses Ctrl C in a CLI environment.

WaitCommodityEvent() would wait for events to happen, unlike CommodityEvent(), - this is useful for saving CPU time.

EventLoop

1.18 filterstrings

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

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

Some simple input description strings.

```
"rawkey upstroke a"
```

```
"rawkey -upsroke f1"
```

```
"timer"
```

```
"diskremoved"
```

```
"rawkey leftbutton f2"
```

1.19 class

Class can be any one of the class strings in the table below.

```
Class String
-----
rawkey
timer
diskremoved
diskinserted
```

1.20 qualifier|synonym

Qualifier is one of the qualifier strings from the table below. 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
```