

Commodity

COLLABORATORS

	<i>TITLE :</i> Commodity		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 26, 2024	

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	3
1.6	changecommodityfilter	3
1.7	changecommodityfilterix	3
1.8	changecommoditytranslator	4
1.9	commodityctrlsignal	4
1.10	commodityevent	4
1.11	commodityid	5
1.12	commoditysignal	5
1.13	commoditytype	5
1.14	createcommodityobject	6
1.15	freecommodityobject	6
1.16	initcommodity	7
1.17	waitcommodityevent	8
1.18	filterstrings	8
1.19	class	9
1.20	qualifierlsynonym	9
1.21	upstroke	9
1.22	highmaplansicode	10
1.23	eventloop1	10
1.24	eventloop2	10

Chapter 1

Commodity

1.1 Commodity V1.00

Pure Basic - Commodity V1.00

A Commodity is a way to manage an application under the AmigaOS. This program must follow established rules to become a commodity: its window can be hidden/showed, the application can be disabled,... All these actions can be done via the 'Exchange' program. A commodity can have several advantages: HotKeys, Break C signals and more.

Commands summary:

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

SYNTAX

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

STATEMENT

Enable or Disable the Commodity, which include all created Objects.

When the Commodity is disabled it will only receive CxMessages of Command Type, from Commodities Exchange, CxMessages of Event Type are not processed.

Status

Enable the Commodity by setting status to TRUE and Disable Commodity by setting it to FALSE.

1.3 activatecommodityobject

SYNTAX

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

STATEMENT

Disable or Enable an Object created by CreateCommodityObject().

A disabled Object is kind of sleeping, it won't process any CxMessages until ActivateCommodityObject(#Obj, TRUE) wakes it up.

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

#Obj

Object to Disable or Enable.

Status

Disable the Object by setting status to FALSE and Enable Object by setting status to TRUE.

1.4 activatecommoditytranslator

SYNTAX

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

STATEMENT

Disable or Enable the Translator in an Object.

An enabled Translator changes the CxMessage input event in two ways, it could be eliminated or replaced by a new input event. When none of this is useful just disable the Translator.

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

#Obj

Object to use.

Status

Disable the Translator by setting status to FALSE and Enable Translator by setting status to TRUE.

1.5 addcommodityinputevent

SYNTAX

AddCommodityInputEvent(*InputEvent)

STATEMENT

Add a new InputEvent or a chain of InputEvents to the input eventstream.

*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.

1.6 changecommodityfilter

SYNTAX

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

FUNCTION

Change the Filter conditions for an Object.

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

#Obj

The Object to change the Filter for.

Filter\$

The string that describes the new Filter conditions.

Result

If it's TRUE the input description string is bad and the Object does not process any CxMessages until either: this function, or ChangeCommodityFilterIX() succeeds the next time.

1.7 changecommodityfilterix

SYNTAX

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

FUNCTION

Change the Filter conditions with an InputXpression Structure.

This function 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 function, or ChangeCommodityFilter() succeeds the next time.

1.8 changecommoditytranslator

SYNTAX

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

SYNTAX

Result.w = CommodityCtrlCSignal()

FUNCTION

When a Commodity event has occurred this function 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

SYNTAX

Result.w = CommodityEvent()

FUNCTION

This function 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

SYNTAX

Result.w = CommodityID()

FUNCTION

This function 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

SYNTAX

Result.w = CommoditySignal()

FUNCTION

When a Commodity event has occurred this function 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

SYNTAX

Result.w = CommodityType()

FUNCTION

This function 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

SYNTAX

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

FUNCTION

This function 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 function 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

SYNTAX

```
FreeCommodityObject (#Obj.l)
```

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

SYNTAX

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

FUNCTION

This function 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

SYNTAX

WaitCommodityEvent()

STATEMENT

This function 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
```