commodities

COLLABORATORS			
	TITLE : commodities		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 29, 2025	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

commodities

1.1 commodities.doc

```
    ActivateCxObj()
    AddIEvents()
    AttachCxObj()
    ClearCxObjError()
    CreateCxObj()
    CxBroker()
    CxMsgData()
    CxMsgID()
    CxMsgType()
    CxObjError()
    CxObjType()
    DeleteCxObj()
    DeleteCxObjAll()
    DisposeCxMsg()
    DivertCxMsg()
    EnqueueCxObj()
    InsertCxObj()
    InvertKeyMap()
    MatchIX()
    ParseIX()
    RemoveCxObj()
    RouteCxMsg()
    SetCxObjPri()
    SetFilter()
    SetFilterIX()
    SetTranslate()
```

1.2 commodities.library/ActivateCxObj

```
  NAME
ActivateCxObj -- change the activation state of a commodity object.
      (V36)

  SYNOPSIS
previous = ActivateCxObj(co,true);
```

```
D0       A0 D0

LONG ActivateCxObj(CxObj *,LONG);

 FUNCTION
Commodity objects of all types maintain an activation state.
If an object is "active", then it performs its particular action
whenever a commodity message arrives. If the object is "inactive"
no action is taken, and the message goes on to its next destination.

All objects are created in the "active" state except brokers
which are created "inactive". Thus, after you create your
broker and hang a tree of objects off of it, you must remember
to use this function to activate it. This causes it to divert all
messages to your tree of objects.

This function activates 'co' if 'true' is different than 0. and
deactivates it otherwise. The previous activation state is
returned.

 INPUTS
co - the commodity object to affect (may be NULL)
true - 0 to deactivate the object, anything else to activate it

 RESULTS
previous - the previous activation state: 0 if the object was
     inactive or if 'co' was NULL, anything else if the object
     was active

 SEE ALSO
CxBroker()
```

1.3 commodities.library/AddIEvents

```
 NAME
AddIEvents -- add input events to commodities' input stream. (V36)

 SYNOPSIS
AddIEvents(events)
     A0

VOID AddIEvents(struct InputEvent *);

 FUNCTION
This function adds a null-terminated linked list of input events to
the input stream processed by commodities. It is a touch easier than
using the input device directly.

The contents of the input events are copied into commodity messages,
so they may be disposed of as soon as this call returns.

The messages are initially routed to the first broker in
commodities' object list.

 INPUTS
```

```
events - the list of input events to add (may be NULL)

 WARNING
The cx_lib/InvertString() function creates lists of input events that
are in reverse order. Thus, passing the result of InvertString()
to this function will insert a series of input events that will
generate a string that's a mirror image of the string passed to
InvertString() originally.

The solution to the above is to either flip the string before
passing it to InvertString(), or flip the resulting list of input
events.

 BUGS
Prior to V40, this function did not copy the data pointed to by
ie_EventAddress for any events of type IECLASS_NEWPOINTERPOS.

 SEE ALSO
cx_lib/FreeIEvents()
```

1.4 commodities.library/AttachCxObj

```
 NAME
AttachCxObj -- attach a commodity object to the end of an existing
         list of objects. (V36)

 SYNOPSIS
AttachCxObj(headObj,co);
      A0      A1

VOID AttachCxObj(CxObj *,CxObj *);

 FUNCTION
Adds 'co' to the list of objects pointed to by 'headObj'. The
new object is added at the end of the list.

 INPUTS
headObj - pointer to a list of objects. If this value is NULL, then
    the entire tree of objects pointed to by 'co' is deleted
    and becomes invalid.
co - the object to add to the list (may be NULL)

 RESULTS
If 'co' is NULL, this function will record that fact in the
internal accumulated error of 'headObj'. This error record
can be retrieved using CxObjError() and cleared using
ClearCxObjError().

 BUGS
Until V38, passing a NULL 'headObj' parameter would cause low-memory
access and unpredictable results.

 SEE ALSO
CxObjError(), ClearCxObjError()
```

1.5 commodities.library/ClearCxObjError

```
 NAME
ClearCxObjError -- clear the accumulated error value of a commodity
       object. (V36)

 SYNOPSIS
ClearCxObjError(co);
                A0

VOID ClearCxObjError(CxObj *);

 FUNCTION
This function clears the accumulated error value of commodity object
'co'.

It is unwise to do this to a filter if COERR_BADFILTER is set.
This will fool commodities into thinking the filter is OK. Set
another valid filter, or leave the error value alone.

 INPUTS
co - the object to affect (may be NULL)

 SEE ALSO
CxObjError()
```

1.6 commodities.library/CreateCxObj

```
 NAME
CreateCxObj -- create a new commodity object. (V36)

 SYNOPSIS
co = CreateCxObj(type,arg1,arg2);
D0     D0   A0   A1

CxObj *CreateCxObj(ULONG,LONG,LONG);

 FUNCTION
This function creates a commodity object of type 'type'. It is not
proper to call this function directly. Each object creation routine
except CxBroker() is defined as a macro in <libraries/commodities.h>
These are independently documented.

All functions which operate on a commodity object are made with
a reference to the thirty-two bit value returned by this function
(or by CxBroker()).

 INPUTS
type - the type of object to create as defined in
       <libraries/commodities.h>
arg1 - first argument, meaning depends on 'type'
arg2 - second argument, meaning depends on 'type'

 RESULTS
```

```
co - a pointer to the new object or NULL if it could not be created.
     A NULL return indicates typically a lack of system memory. Minor
     problems in creating an object, such as providing a bad filter
     description to cx_lib/CxFilter(), typically don't cause failure,
     but are recorded in an internal error field in the new object
     which can be accessed via CxObjError().

 SEE ALSO
CxObjError(), cx_lib/CxFilter(), cx_lib/CxSender(),
cx_lib/CxSignal(), cx_lib/CxTranslate(), cx_lib/CxDebug(),
cx_lib/CxCustom(), CxBroker()
```

1.7 commodities.library/CxBroker

```
 NAME
CxBroker -- create a commodity broker. (V36)

 SYNOPSIS
broker = CxBroker(nb,error);
D0      A0 D0

CxObj *CxBroker(struct NewBroker *,LONG *);

 FUNCTION
This function creates a broker from the specification found in the
NewBroker structure pointed to by 'nb'. The purpose and meaning of the
fields of the NewBroker structure are described below. Also see
<libraries/commodities.h> for more info.

struct NewBroker
{
    BYTE            nb_Version;
    STRPTR          nb_Name;
    STRPTR          nb_Title;
    STRPTR          nb_Descr;
    WORD            nb_Unique;
    WORD      nb_Flags;
    BYTE            nb_Pri;
    struct MsgPort *nb_Port;
    WORD            nb_ReservedChannel;
};

nb_Version
This is the way that future versions of commodities can identify
which version of the NewBroker structure you are using. This should be
set to NB_VERSION as defined in <libraries/commodities.h>

nb_Name
This is the name of the broker. This name is used to find the broker
in commodities' object list and is the name shown in the listview
gadget of the Exchange program. The name string is copied in the
broker object upon creation so it can be discarded right after
CxBroker() returns. The maximum length of the name string is defined
by a constant in <libraries/commodities.h>.
```

```
nb_Title, nb_Descr
These are two strings which appear to the user in the Exchange
program and describe the application the broker is representing.
Note that these strings are copied into the broker object so they
can be discarded right after CxBroker() returns. The maximum length
of these strings that will be recognized are defined by constants in
<libraries/commodities.h>.

nb_Unique
This field indicates what should happen if a broker of the same name
(nb_Name) already exists in commodities' object list. Constants
in <libraries/commodities.h> allow the caller to specify whether
another broker is to be created, and whether any existing broker of
the same name should be notified that an attempt at creating a
duplicate has been made.

nb_Pri
This specifies with what priority the new broker is to be inserted
within commodities' object list. Higher priority nodes appear
earlier in a list. It is strongly recommended that the ToolTypes
environment of an application be used to allow the end-user to set
the priority of the broker.

 INPUTS
nb - an initialized NewBroker structure
error - a pointer to a longword where to store a failure code (may be
  NULL)

 RESULTS
broker - a pointer to the broker object or NULL upon failure. If the
   'error' pointer is not NULL, a further diagnostic code is
   placed at that address. Error codes are defined in
   <libraries/commodities.h> and include:

    CBERR_OK
    No problems; broker created OK.

    CBERR_SYSERR
    System problems, not your fault, sign of low memory.

    CBERR_DUP
    The nb_Unique field specified that only one broker of
    'nb_Name' should be allowed, and one already exists.

    CBERR_VERSION
    The version specified in 'nb_Version' is unknown to the
    library.

 SEE ALSO
SetCxObjPri(), <libraries/commodities.h>
```

1.8 commodities.library/CxMsgData

```
 NAME
CxMsgData -- obtain a pointer to a commodity message's data area. (V36)
```

```
 SYNOPSIS
data = CxMsgData(cxm);
D0     A0

APTR CxMsgData(struct CxMsg *);

 FUNCTION
Most commodity messages contain meaningful data such as an InputEvent
structure. This function returns a pointer to this data.

You may get a commodity message from a synchronous (custom object) or
asynchronous (sender object) source. In the second case, 'data' is
not valid after you have replied to the message.

 INPUTS
cxm - the commodity message to get the data pointer from (may be NULL)

 RESULTS
data - a pointer to the message's data, or NULL if 'cxm' is NULL.
       The meaning of the data varies depending on which kind of
       object is being inspected.

 BUGS
Until V38, passing a NULL 'cxm' parameter would cause low-memory
access and unpredictable results.

Until V40, whenever the data pointer is a (struct InputEvent), the
ie_EventAddress field of these structures was not reliable whenever
the message was generated by a sender object.

 SEE ALSO
cx_lib/CxSender(), cx_lib/CxCustom()
```

1.9 commodities.library/CxMsgID

```
 NAME
CxMsgID -- obtain the ID of a commodity message. (V36)

 SYNOPSIS
id = CxMsgID(cxm);
D0       A0

LONG CxMsgID(struct CxMsg *);

 FUNCTION
This function returns the value associated with the cause or source
of the commodity message 'cxm'. Values are provided by the application
when a sender or custom object is created.

 INPUTS
cxm - the commodity message to inquire about (must NOT be NULL)

 RESULTS
id - if not specified by the application, the ID value of a
```

```
      commodity message will be 0. It is suggested that using
      non-zero values in your program as a rule may identify some
      possible errors.

 SEE ALSO
cx_lib/CxSender(), cx_lib/CxCustom()
```

1.10 commodities.library/CxMsgType

```
 NAME
CxMsgType -- obtain the type of a commodity message. (V36)

 SYNOPSIS
type = CxMsgType(cxm);
D0     A0

ULONG CxMsgType(struct CxMsg *);

 FUNCTION
This function returns the type of a commodity message. Possible values
of 'type' are defined in <libraries/commodities.h>. Most commodity
messages are of type CXM_IEVENT.

 INPUTS
cxm - the commodity message to inquire about (must NOT be NULL)

 RESULTS
  type - the type of the commodity message, possible values are defined
       in <libraries/commodities.h>
```

1.11 commodities.library/CxObjError

```
 NAME
CxObjError -- obtain a commodity object's accumulated error. (V36)

 SYNOPSIS
error = CxObjError(co);
D0       A0

LONG CxObjError(CxObj *);

 FUNCTION
When a function acting on an object fails, it records the failure
in the object. This function returns the accumulated error value.
The values are represented by flag bits defined in
<libraries/commodities.h>. Several errors may be recorded by multiple
bits in 'error'.

The currently defined errors are:

COERR_ISNULL
  The value of parameter 'co' was in fact NULL. This error
```

```
   means "the problem with the object you inquire about is
   that it failed to be created."

   COERR_NULLATTACH
   Using the commodities' list manipulation functions, an attempt was
   made to add a NULL object to the list belonging to 'co'. This allows
   a line of code as follows to exist in an error-tolerant program:

     AttachCxObj(filter,CxSender(myport,MY_ID));

   COERR_BADFILTER
   The most recent filter specification for a filter object
   was faulty. This happens if no sense can be made out of a
   description string, or if an input expression (IX) has an
   invalid format or version byte. When this bit is set in a
   filter's error field, the filter will match nothing, but this
   is not the proper way to "turn off" a filter, use ActivateCxObj().

   COERR_BADTYPE
   A type specific operation, such as SetFilterIX(), was called
   for object 'co', but 'co' isn't of the proper type.

 INPUTS
co - the commodity object to get the accumulated error from (may be
     NULL)

 RESULTS
error - the accumulated error, or 0 if 'co' is NULL

 SEE ALSO
SetFilter(), SetFilterIX(), AttachCxObj(), ActivateCxObj(),
ClearCxObjError()
```

1.12 commodities.library/CxObjType

```
 NAME
CxObjType -- obtain the type of a commodity object. (V36)

 SYNOPSIS
type = CxObjType(co);
D0     A0

ULONG CxObjType(CxObj *);

 FUNCTION
This function should not really be necessary. It returns
the type of a commodity object, which you should already
know, since you created it in the first place.

 INPUTS
co - the commodity object to inquire about (may be NULL)

 RESULTS
type - the type of the commodity object, possible values are
       defined in <libraries/commodities.h>. Returns CX_INVALID
```

```
       if 'co' is NULL.

 SEE ALSO
CreateCxObj()
```

1.13 commodities.library/DeleteCxObj

```
 NAME
DeleteCxObj -- delete a commmodity object. (V36)

 SYNOPSIS
DeleteCxObj(co);
            A0

VOID DeleteCxObj(CxObj *);

 FUNCTION
Deletes a commodity object of any type. If the object
is linked into a list, it is first removed. Note that
the handle 'co' is invalid after this function is called.

Also note that deleting an object which has other objects attached
to it may be undesirable. Use the function DeleteCxObjAll()
to delete an entire sub-tree of objects.

 INPUTS
co - the commodity object to delete (may be NULL)

 SEE ALSO
DeleteCxObjAll()
```

1.14 commodities.library/DeleteCxObjAll

```
 NAME
DeleteCxObjAll -- recursively delete a tree of commodity objects.
      (V36)

 SYNOPSIS
DeleteCxObjAll(co);
         A0

VOID DeleteCxObjAll(CxObj *);

 FUNCTION
This function deletes the commodity object 'co', and
recursively deletes all objects attached to it, and the
objects attached to them, etc.

If 'co' is linked into a list, it is first removed. Note
that the handle 'co' is invalid after this function is called.

This function is useful when an application exits: most
```

```
applications can clean up completely by deleting the
entire sub-tree of objects starting at their broker.

 INPUTS
co - the first commodity object to delete (may be NULL)

 SEE ALSO
DeleteCxObj()
```

1.15 commodities.library/DisposeCxMsg

```
 NAME
DisposeCxMsg -- delete a commodity message. (V36)

 SYNOPSIS
DisposeCxMsg(cxm);
       A0

VOID DisposeCxMsg(struct CxMsg *);

 FUNCTION
This function eliminates the commodity message pointed to by 'cxm'.
Can be used to 'swallow' input events by disposing of every commodity
message of type CXM_IEVENT.

This function can only be called from within a custom object running
on the input handler's context. It cannot be called from code running
on a commodities' context, such as when receiving a CXM_IEVENT message
from a sender object. CxMsg sent to a commodity program from a sender
object must be sent back using ReplyMsg().

 INPUTS
cxm - the commodity message to delete (must NOT be NULL)
```

1.16 commodities.library/DivertCxMsg

```
 NAME
DivertCxMsg -- send a commodity message down an object list. (V36)

 SYNOPSIS
DivertCxMsg(cxm,headObj,returnObj);
      A0  A1      A2

VOID DivertCxMsg(struct CxMsg *,CxObj *,CxObj *);

 FUNCTION
This function sends the commodity message 'cxm' down the list of
objects attached to 'headObj'. The pointer 'returnObj' is first
pushed onto the routing stack of 'cxm' so that when the end of the
list of 'headObj' is reached the SUCCESSOR of 'returnObj' is the
next destination.
```

```
For example, when a filter finds a match with a message, the
message is diverted down the filter's list like this:

  DivertCxMsg(cxm,filter,filter);

 INPUTS
cxm - the commodity message to divert (must NOT be NULL)
headObj - the list of objects to divert the message to
returnObj - the object to use as a place holder

 SEE ALSO
RouteCxMsg()
```

1.17 commodities.library/EnqueueCxObj

```
 NAME
EnqueueCxObj -- insert a commodity object within a list of objects
          based on object priority. (V36)

 SYNOPSIS
EnqueueCxObj(headObj,co);
       A0      A1

VOID EnqueueCxObj(CxObj *,CxObj *);

 FUNCTION
This function puts object 'co' into the list of object 'headObj'.
The insertion point is determined by the object's priority. The
objects are kept in the list from the highest priority to the
lowest. New nodes are inserted in front of the first node with a
lower priority. Hence a FIFO queue for nodes of equal priority.

The priority of the commodity object can be set using SetCxObjPri().

 INPUTS
headObj - pointer to a list of objects. If this value is NULL, then
    the entire tree of objects pointed to by 'co' is deleted
    and becomes invalid.
co - the object to add to the list (may be NULL)

 RESULTS
If 'co' is NULL, this function will record that fact in the
internal accumulated error of 'headObj'. This error record
can be retrieved using CxObjError() and cleared using
ClearCxObjError().

 BUGS
Until V38, passing a NULL 'headObj' parameter would cause low-memory
access and unpredictable results.

 SEE ALSO
SetCxObjPri(), CxObjError(), ClearCxObjError()
```

1.18 commodities.library/InsertCxObj

```
 NAME
InsertCxObj -- insert a commodity object in a list after a given
         object. (V36)

 SYNOPSIS
InsertCxObj(headObj,co,pred);
      A0      A1 A2

VOID InsertCxObj(CxObj *,CxObj *,CxObj *);

 FUNCTION
Adds 'co' to the list of objects pointed to by 'headObj' after
object 'pred'.

 INPUTS
headObj - pointer to a list of objects. If this value is NULL, then
    the entire tree of objects pointed to by 'co' is deleted
    and becomes invalid.
co - the object to add to the list (may be NULL)
pred - the object after which 'co' should be inserted. If this is NULL
       then 'co' is added to the head of the list.

 RESULTS
If 'co' is NULL, this function will record that fact in the
internal accumulated error of 'headObj'. This error record
can be retrieved using CxObjError() and cleared using
ClearCxObjError().

 BUGS
Until V38, passing a NULL 'headObj' parameter would cause low-memory
access and unpredictable results.

 SEE ALSO
CxObjError(), ClearCxObjError()
```

1.19 commodities.library/InvertKeyMap

```
 NAME
InvertKeyMap -- generate an input event from an ANSI code. (V36)

 SYNOPSIS
success = InvertKeyMap(ansiCode,event,km)
D0           D0       A0    A1

BOOL InvertKeyMap(ULONG,struct InputEvent *,struct KeyMap *);

 FUNCTION
This function uses the system call MapANSI() to figure out what
InputEvent translates to an ANSI character code 'ansiCode'. The
InputEvent pointed to by 'event' is filled in with that information.
The KeyMap 'km' is used for the translation, unless it is NULL, in
which case the current system default keymap is used.
```

```
 INPUTS
ansiCode - the ANSI code to convert to an input event
event - the InputEvent to fill-in
km - the keymap to use for the translation, or NULL to use the
     current system default keymap.

 RESULTS
success - TRUE if the translation worked, FALSE otherwise.

 BUGS
This function currently handles one-deep dead keys (such as
<alt f>o ). It does not look up the high key map (keystrokes
with scan codes greater than 0x40).

Prior to V40, this function was not initializing the ie_SubClass
and ie_TimeStamp fields of the InputEvent structure. A simple work
around to the problem is to clear these values to 0 before making
a call to this function:

  struct InputEvent ie;

      ie.ie_SubClass           = 0;
      ie.ie_TimeStamp.tv_secs  = 0;
      ie.ie_TimeStamp.tv_micro = 0;
      if (InvertKeyMap(ansiCode,&ie,NULL))
      ...

 SEE ALSO
cx_lib/InvertString()
```

1.20 commodities.library/MatchIX

```
 NAME
MatchIX -- see if an input event matches an initialized input
     expression. (V38)

 SYNOPSIS
match = MatchIX(event,ix);
D0    A0    A1

BOOL MatchIX(struct InputEvent *,IX *);

 FUNCTION
This function determines whether an input event matches an
initialized input expression. Applications generally do not
need to call this function as filter objects will normally provide
all the event filtering needed. Nevertheless, MatchIX()
can come in handy as it is the same function used to match an
event to the various filter objects when an event makes its way
through the input network.

 INPUTS
event - the input event to match against the input expression
ix - the input expression for the comparison
```

```
 RESULTS
match - TRUE if the input event matches the input expression, or
  FALSE if not

 SEE ALSO
<libraries/commodities.h>, ParseIX()
```

1.21 commodities.library/ParseIX

```
 NAME
ParseIX -- initialize an input expression given a description string.
     (V36)

 SYNOPSIS
failureCode = ParseIX(description,ix);
D0          A0          A1

LONG ParseIX(STRPTR,IX *);

 FUNCTION
Given an input description string and an allocated input
expression, sets the fields of the input expression to
correspond to the description string.

 INPUTS
description - the string to parse
ix - the input expression to hold the result of the parse

 RESULTS
failureCode -  0 if all went well,
        -1 if tokens after end (code spec)
              -2 if 'description' was NULL

 SEE ALSO
<libraries/commodities.h>, MatchIX()
```

1.22 commodities.library/RemoveCxObj

```
 NAME
RemoveCxObj -- remove a commodity object from a list. (V36)

 SYNOPSIS
RemoveCxObj(co);
      A0

VOID RemoveCxObj(CxObj *);

 FUNCTION
This function removes 'co' from any list it may be a part of.
Will not crash if 'co' is NULL, or if it has not been inserted
in a list.
```

```
 INPUTS
co - the object to remove (may be NULL)

 SEE ALSO
AttachCxObj(), EnqueueCxObj(), InsertCxObj()
```

1.23 commodities.library/RouteCxMsg

```
 NAME
RouteCxMsg -- set the next destination of a commodity message. (V36)

 SYNOPSIS
RouteCxMsg(cxm,co);
     A0  A1

VOID RouteCxMsg(struct CxMsg *,CxObj *);

 FUNCTION
Establishes the next destination of a commodity message
to be 'co', which must be a valid commodity object, and
must be linked in ultimately to commodities' object list.

Routing of an object is analogous to a 'goto' in a program.
There is no effect on the message's routing stack.

 INPUTS
cxm - the commodity message to route (must NOT be NULL)
co - the commodity object to route the message to (must NOT be NULL)

 SEE ALSO
DivertCxMsg()
```

1.24 commodities.library/SetCxObjPri

```
 NAME
SetCxObjPri -- set the priority of a commodity object. (V36)

 SYNOPSIS
oldPri = SetCxObjPri(co,pri)
D0                   A0 D0

LONG SetCxObjPri(CxObj *,LONG);

 FUNCTION
This function sets the priority of a commodity object for the
purposes of EnqueueCxObj().

It is strongly recommended that the ToolTypes environment be
utilized to provide end-user control over the priority of
brokers, but application specific ordering of other objects
within their lists is not dictated.
```

```
 INPUTS
co - the commodity object to affect (may be NULL)
pri - the object's new priority in the range -128 to +127. A value
      of 0 is normal.

 RESULTS
oldPri - the previous priority of the object or 0 if 'co' was NULL.
   This value is only returned in V38 and beyond.

 BUGS
This function will not reposition an object within its list when
its priority changes. To attain the same effect, first remove the
object from its list using RemoveCxObj(), set its priority using
SetCxObjPri(), and reinsert it in the list using EnqueueCxObj().

 SEE ALSO
EnqueueCxObj()
```

1.25 commodities.library/SetFilter

```
 NAME
SetFilter -- change the matching condition of a commodity filter.
       (V36)

 SYNOPSIS
SetFilter(filter,text);
    A0     A1

VOID SetFilter(CxObj *,STRPTR);

 FUNCTION
This function changes the matching condition of a commodity input
filter to that described by the input description string 'text'.

 INPUTS
filter - the filter object to affect (may be NULL)
text - the new matching conditions for the filter

 RESULTS
The internal error of 'filter' will have the COERR_BADFILTER
bit set or cleared depending on the failure or success of this
function.

 SEE ALSO
SetFilterIX(), CxObjError()
```

1.26 commodities.library/SetFilterIX

```
 NAME
SetFilterIX -- change the matching condition of a commodity filter.
         (V36)
```

```
 SYNOPSIS
SetFilterIX(filter,ix);
      A0     A1

VOID SetFilterIX(CxObj *,IX *);

 FUNCTION
This function changes the matching condition of a commodity input
filter to that described by the binary input expression pointed by
'ix'.

Input expressions are defined in <libraries/commodities.h>.
It is important to remember that the first field of the input
expression structure must indicate which version of the
input expression structure is being used.

 INPUTS
filter - the filter object to affect (may be NULL)
ix - the new matching conditions for the filter

 RESULTS
The internal error of 'filter' will have the COERR_BADFILTER
bit set or cleared depending on the failure or success of this
function.

 SEE ALSO
SetFilter(), CxObjError()
```

1.27 commodities.library/SetTranslate

```
 NAME
SetTranslate -- replace a translator object's translation list. (V36)

 SYNOPSIS
SetTranslate(translator,events);
       A0         A1

VOID SetTranslate(CxObj *,struct InputEvent *);

 FUNCTION
This function replaces the translation list of a commodity
translator object with the linked list starting at 'events'.

A NULL value for 'events' indicates that the object 'translator'
should swallow all commodity messages that are sent its way.

Note that the input events are not copied into commodities' private
memory, but the value of 'events' is used -- asynchronously to the
application program -- to find a chain of InputEvents in the
application's data space. At the time of translation, each input event
is copied into its own new commodity message.

The above means that no other commodities' user, nor
commodities.library itself will be modifying your list of InputEvents.
```

On the other hand, your program must not corrupt the input event chain that has been presented to a translator.

INPUTS

translator - the translator object to affect (may be NULL)
events - the new input event translation list

BUGS

The list of input events manipulated by a translator object is inserted in reverse order in the commodities network, and come out of the network in reverse order as well. The cx_lib/InvertString() function creates lists of input events that are in reverse order so they can be used directly with translator objects.

SEE ALSO

<devices/inputevent.h>, cx_lib/CxTranslate()