

```

/* broker.c - Simple skeletal example of opening a broker
 * compiled with SASC 5.10
 * LC -b0 -cfist -v broker.c
 * Blink FROM LIB:c.o,broker.o TO broker LIBRARY
 * LIB:LC.lib,LIB:Amiga.lib
 */
#include <exec/libraries.h>
#include <libraries/commodities.h>
#include <dos/dos.h>
#include <clib/exec_protos.h>
#include <clib/alib_protos.h>
#include <clib/alib_stdio_protos.h>
#include <clib/alib_commodities_protos.h>
#include <clib/commodities_protos.h>

#ifdef LATTICE
int CXBRK(void) { return(0); } /* Disable Lattice CTRL/C handling */
int chKabort(void) { return(0); }
#endif

struct Library *CxBBase;

void main(void);
LONG ProcessMsg(void);

CXObject *broker;

struct MsgPort *broker_mp;

struct NewBroker newbroker = {
    NB_VERSION, /* nb_Version - Version of the NewBroker structure */
    "AmigaMail broker", /* nb_Name - Name CX uses to identify this commodity */
    "Broker", /* nb_Title - Title of commodity that appears in CXExchange */
    "A simple example of a broker", /* nb_Descr - Description of the commodity */
    0, /* nb_Unique - Tells CX not to launch new commodity with same name */
    0, /* nb_Flags - Tells CX if this commodity has a window */
    0, /* nb_Pri - This commodity's priority */
    0, /* nb_Port - MsgPort CX talks to */
    0 /* nb_ReservedChannel - reserved for later use */
};

ULONG cxsigflag;

void main()
{
    /* Before bothering with anything else, open the library */
    if (CxBBase = OpenLibrary("commodities.library", 37L))
    {
        /* Commodities talks to a Commodities application through
         * an Exec Message port, which the application provides
         */
        if (broker_mp = CreateMsgPort())
        {
            newbroker.nb_Port = broker_mp;

            /* The commodities.library function CxBroker() adds a
             * broker to the master list. It takes two arguments,
             * a pointer to a NewBroker structure and a pointer to
             * a LONG. The NewBroker structure contains information
             * to set up the broker. If the second argument is not
             * NULL, CxBroker will fill it in with an error code.
             */
            if (broker = CxBroker(&newbroker, NULL))
            {
                cxsigflag = 1L << broker_mp->mp_SigBit;

                /* After it's set up correctly, the broker
                 * has to be activated
                 */
                ActivateCXObject(broker, 1L);

                /* the main processing loop */
                while (ProcessMsg());

                /* It's time to clean up. Start by removing

```

```

         * the broker from the Commodities master list.
         * The DeleteCXObjectAll() function will take care
         * of removing a CXObject and all those connected
         * to it from the Commodities network
         */
        DeleteCXObject(broker);
    }
    DeleteMsgPort(broker_mp);
}
CloseLibrary(CxBBase);
}

LONG ProcessMsg(void)
{
    CxMsg *msg;

    ULONG sigrcvd, msgid, msgtype;
    LONG returnvalue = 1L;

    /* wait for something to happen */
    sigrcvd = Wait(SIGBREAKF_CTRL_C | cxsigflag);

    /* process any messages */
    while(msg = (CxMsg *)GetMsg(broker_mp))
    {
        /* Extract necessary information from the CxMessage and return it */
        msgid = CxMsgID(msg);
        msgtype = CxMsgType(msg);
        ReplyMsg((struct Message *)msg);

        switch(msgtype)
        {
            case CXM_IEVENT:
                /* Shouldn't get any of these in this example */
                break;
            case CXM_COMMAND:
                /* Commodities has sent a command */
                printf("A command: ");
                switch(msgid)
                {
                    case CXCMD_DISABLE:
                        printf("CXCMD_DISABLE\n");
                        /* The user clicked Commodities Exchange disable
                         * gadget better disable
                         */
                        ActivateCXObject(broker, 0L);
                        break;
                    case CXCMD_ENABLE:
                        /* user clicked enable gadget */
                        printf("CXCMD_ENABLE\n");
                        ActivateCXObject(broker, 1L);
                        break;
                    case CXCMD_KILL:
                        /* user clicked kill gadget, better quit */
                        printf("CXCMD_KILL\n");
                        returnvalue = 0L;
                        break;
                }
                break;
            default:
                printf("Unknown msgtype\n");
                break;
        }
    }

    /* Test to see if user tried to break */
    if (sigrcvd & SIGBREAKF_CTRL_C)
    {
        returnvalue = 0L;
        printf("CTRL C signal break\n");
    }
    return(returnvalue);
}

```

## Input

## Introduction to Commodities Exchange

```

/* Hockey.c - simple hot key commodity
 * compiled with SASC 5.10
 * IC -b0 -offset -v hockey.c
 * Blink FROM LIB:c.o,hockey.o TO hockey LIBRARY LIB:LC.lib,LIB:Amiga.lib
 */
#include <exec/libraries.h>
#include <libraries/commodities.h>
#include <dos/dos.h>
#include <clib/exec_protos.h>
#include <clib/alib_protos.h>
#include <clib/alib_stdio_protos.h>
#include <clib/commodities_protos.h>

#define LATTICE
int CxSRK(void) { return(0); } /* Disable Lattice CTRL/C handling */
#define f

#define EVT_HOTKEY IL

struct Library *CxBase, *IconBase;

void main(int, char **);
LONG ProcessMsg(void);

struct MsgPort *broker_mp;

CxBj *broker, *filter, *sender, *translate;

struct NewBroker newbroker = {
    NB_VERSION,          /* string to identify this broker */
    "AmiMail Hockey",
    "A Simple Hockey",
    NBU_UNIQUE | NBU_NOTIFY, /* Don't want any new commodities
                             * starting with this name. If someone
                             * tries it, let me know */
    0,
    0,
    0,
    0,
};

ULONG cxsigflag;

void main(int argc, char **argv)
{
    char *hockey, **types;

    if (CxBase = OpenLibrary("commodities.library", 37L))
    {
        /* open the icon.library for the support library
         * functions, ArgArrayInit() and ArgArrayDone()
         */
        if (IconBase = OpenLibrary("icon.library", 36L))
        {
            if (broker_mp = CreateMsgPort())
            {
                newbroker.nb_port = broker_mp;
                cxsigflag = IL << broker_mp->mp_sigflag;

                /* ArgArrayInit() is a support library function
                 * (from the 2.0 version of amiga.lib) that makes it
                 * easy to read arguments from either a CLI or from
                 * the Workbench's ToolTypes. Because it uses
                 * icon.library, the library has to be open before
                 * calling this function. ArgArrayDone() cleans up
                 * after this function.
                 */
                types = ArgArrayInit(argc, argv);

                /* ArgInt() (also from amiga.lib) searches through the
                 * array set up by ArgArrayInit() for a specific

```

```

* ToolType. If it finds one, it returns the numeric
* value of the number that followed the ToolType
* (i.e. CX_PRIORITY=7). If it doesn't find the ToolType,
* it returns the default value (the third argument)
*/
newbroker.nb_pri = (BYTE)ArgInt(types, "CX_PRIORITY", 0);

/* ArgString() works just like ArgInt(), except it
 * returns a pointer to a string rather than an integer.
 * In the example below, if there is no ToolType "HOTKEY",
 * the function returns a pointer to "rakey control esc".
 */
hockey = ArgString(types, "HOTKEY", "rakey control esc");

if (broker = CxBroker(&newbroker, NULL))
{
    /* CxFilter() is a macro that creates a filter
     * CXObject. This filter passes input events that
     * match the string pointed to by hockey.
     */
    if (filter = CxFilter(hockey))
    {
        /* Add a CXObject to another's personal list */
        AttachCxBj(broker, filter);

        /* CxSender() creates a sender CXObject. Every
         * time a sender gets a CxMessage, it sends a new
         * CxMessage to the port pointed to in the first
         * argument. CxSender()'s second argument will be
         * the ID of any CxMessages the sender sends to
         * the port. The data pointer associated with the
         * CxMessage will point to a *COPY* of the
         * InputEvent structure associated with the original
         * CxMessage.
         */
        if (sender = CxSender(broker_mp, EVT_HOTKEY))
        {
            AttachCxBj(filter, sender);

            /* CxTranslate() creates a translate CXObject.
             * When a translate CXObject gets a CxMessage,
             * it deletes the original CxMessage and adds
             * a new input event to the input.device's
             * input stream after the Commodities input
             * handler. CxTranslate's argument points
             * to an InputEvent structure from which to
             * create the new input event. In this example,
             * the pointer is NULL, meaning no new event
             * should be introduced.
             */
            if (translate = (CxTranslate(NULL)))
            {
                AttachCxBj(filter, translate);

                /* CxObjError() is a commodities.library
                 * function that returns the internal
                 * accumulated error code of a CXObject.
                 */
                if (! CxObjError(filter))
                {
                    ActivateCxBj(broker, IL);
                    while (ProcessMsg());
                }
            }
        }
        /* DeleteCxBjAll() is a commodities.library function
         * that not only deletes the CXObject pointed to in
         * its argument, but it deletes all of the CxObjects
         * that are attached to it.
         */
        DeleteCxBjAll(broker);
        DeleteMsgPort(broker_mp);
    }
}

```

```

        /* this amiga.lib function cleans up after ArgArrayInit() */
        ArgArrayDone();
        CloseLibrary(IconBase);
    }
    CloseLibrary(CxBase);
}

LONG ProcessMsg(void)
{
    extern struct MsgPort *broker_mp;
    extern CxObj *broker;

    extern ULONG cxsigflag;

    CxMsg *msg;

    ULONG sigrcvd, msgid, msgtype;
    LONG returnvalue = 1L;

    sigrcvd = Wait(SIGBREAKF_CTRL_C | cxsigflag);

    while(msg = (CxMsg *)GetMsg(broker_mp))
    {
        msgid = CxMsgID(msg);
        msgtype = CxMsgType(msg);
        ReplyMsg((struct Message *)msg);

        switch(msgtype)
        {
            case CXM_IEVENT:
                printf("A CXM_EVENT, ");
                switch(msgid)
                {
                    case EVT_HOTKEY:
                        /* We got the message from the sender CxObject */
                        printf("You hit the HotKey.\n");
                        break;

                    default:
                        printf("unknown.\n");
                        break;
                }
                break;

            case CXM_COMMAND:
                printf("A command: ");
                switch(msgid)
                {
                    case CXCMD_DISABLE:
                        printf("CXCMD_DISABLE\n");
                        ActivateCxObj(broker, 0L);
                        break;

                    case CXCMD_ENABLE:
                        printf("CXCMD_ENABLE\n");
                        ActivateCxObj(broker, 1L);
                        break;

                    case CXCMD_KILL:
                        printf("CXCMD_KILL\n");
                        returnvalue = 0L;
                        break;

                    case CXCMD_UNIQUE:
                        /* Commodities Exchange can be told not
                         * only to refuse to launch a commodity with
                         * a name already in use but also can notify
                         * the already running commodity that it happened.
                         * It does this by sending a CXM_COMMAND with the

```

```

                         * ID set to CXMCMD_UNIQUE. If the user tries
                         * to run a windowless commodity that is already
                         * running, the user wants the commodity to shut down.
                        */
                        printf("CXCMD_UNIQUE\n");
                        returnvalue = 0L;
                        break;

                    default:
                        printf("Unknown msgid\n");
                        break;
                }
                break;

            default:
                printf("Unknown msgtype\n");
                break;
        }
    }

    if (sigrcvd & SIGBREAKF_CTRL_C)
    {
        returnvalue = 0L;
        printf("CTRL C signal break\n");
    }

    return(returnvalue);
}

```

