BNCS Class Library

External Infodriver Interface Class

Written by Tim Alden

Overview

This class provides all the required connectivity between your driver application and a BNCS infodriver. It encapsulates all data transfer to and from slots, and provides the required callback mechanism for asynchronous slot change notifications.

The class definition is as follows:

```
class extinfo
{
        HWND hWndSpawn;
        PULONG txcount;
        PULONG rxcount;
        int requestmode;
        ULONG deftx,defrx;
        HINSTANCE hInstEx;
public:
        void (*func)(extinfo*,UINT,LPCSTR);
        UINT iDevice;
        HWND hWndInfo;
        LPSTR netmsg;
        extinfo();
        ~extinfo();
        int iStatus;
        int connect(UINT iEDev,HINSTANCE hInst=NULL,UINT iMyDev=0,UINT iOffs=0,UINT iMin=0,UINT iMax=0);
        void notify(void(*)(extinfo*,UINT,LPCSTR));
        void setcounters(PULONG lpTX,PULONG lpRX);
        void setslot(int, LPCSTR, ...);
        void updateslot(int, LPCSTR, ...);
        void getslot(int, LPSTR );
        void sendslots(int,int);
        void incrx();
        int getrequestmode(){return requestmode;}
        int setmode(int iMode);
        int getmode();
        UINT iMinRange;
        UINT iMaxRange;
        UINT iOffset;
        BOOL fCalledEx;
        UINT iThisDev;
        HWND hWndCSI_Cl;
        void getdbname(WORD wDevice, WORD wDatabase, WORD wIndex, LPSTR szName, int iMaxSize);
        int getdbindex(WORD wDevice, WORD wDatabase, LPCSTR szName);
        void setdbname(WORD device, WORD database, WORD index, LPCSTR name, BOOL fPoll);
};
```

Required files

The class is contained in the library **`extinfo.lib`** and a debug version in **`extinfodb.lib`**

The header files **`<extinfo.h>`** and **`<bncsdef.h>`** should be referenced by your application.

The following constants are defined in the **`<bncsdef.h>`** header file, and are valid values for the `iStatus` member variable:

```
#define ERROR_GENERAL             0xFFFF
#define ERROR_WRONG_TYPE          0xFFFE
#define ERROR_BAD_PARAMLIST       0xFFFD

#define COMMAND                   0
#define STATUS                    1
#define RXONLY                    2
#define DISCONNECTED              3
#define INVALID_DRIVERNUM         4
#define CANT_FIND_INFODRIVER      5
#define TO_TXRX                   6
#define TO_RXONLY                 7
#define QUERY_TXRX                8
#define CANT_REGISTER_CLASSWND    9
#define DRIVERNUM_ALREADY         10
#define CSI_NOT_ENABLED           11
#define DATABASECHANGE            12
#define BAD_WS                    13
#define CONNECTED                 15
```

Usage

Declare an instance of the class extinfo:
```
extinfo ex;
```

Specify the driver number when connecting to the infodriver
```
ex.connect(iInfoExt);
```
or
```
ex.connect(iInfoExt,NULL,iMyHostNum,iOffset,iMin,iMax);
```

Using the first example syntax connects a standard infodriver external. The second example syntax above can be used to make the infodriver external "infohost compliant": connecting to an infohost driver (iInfoExt=1000+) and specifying a unique iMyHostNum *value to identify your connection. You also specify the zero-based slot offset* iOffset *and slot range* iMin *to* iMax*. In this way, many externals can share one infodriver by fanning in through the infohost driver. Please refer to the InfoHost instructions for further information.*

The status/success of the connection to the infodriver can be ascertained by reading the status member of the function:
example:
```
      if (ex.iStatus==CANT_FIND_INFODRIVER)
            Debug("Infodriver not found");
```

Assign the callback function which the class will send slotchange messages to:
```
ex.notify(InfoNotify);
```

where the function is of the form:
```
void SlotChange(extinfo* pex,UINT iSlot,LPCSTR szSlot);
```

sample function:
```
void InfoNotify(extinfo* pex,UINT iSlot,LPCSTR szSlot)
{
        switch (pex->iStatus) {
        case CONNECTED: // this is the "normal" situation
                if (!SlotChange(pex->iDevice,iSlot,szSlot))
                        pex->setslot(iSlot,szSlot);
                break;

        case DISCONNECTED:
                Debug("Infodriver %d has disconnected", pex->iDevice);
                return;

        case TO_RXONLY:
                Debug("Infodriver %d received request to go RX Only",pex->iDevice);
                break;

        case TO_TXRX:
                Debug("Infodriver %d received request to go TXRX",pex->iDevice);
                break;

        default:
                Debug("iStatus=%d",pex->iStatus);

        }

}
```

Always check the status member first in case the class is informing you of a termination, or other special event. A valid slot change notification will always have CONNECTED status!

The connection is closed automatically when the program terminates and the class goes out of scope.

Set the contents of a slot as follows:
```
      ex.setslot(iSlotnum,szContents);
or    ex.setslot(iSlotnum,”Value=%d”,iVal); // e.g. like printf
```

It is possible to set a slot without issuing a network revertive by setting the iSlotnum parameter to be the negative equivalent of the desired slot number, e.g. gpex->setslot(-iSlotnum,szContents);

Update the contents of a slot as follows:
```
      ex.updateslot(iSlotNum,szContents);
or    ex.updateslot(iSlotNum,szFmt, …);
```

Use update to change the slot contents only if they are different to the existing contents. Negative slot numbers work as for setslot().

Retrieve the contents of a slot as follows:
```
ex.getslot(iSlotnum,szContents);
```

Issue a network revertive for a range of slots as follows:
```
ex.sendslots(iMinSlot, iMaxSlot);
```

Set up automatic counters for incoming and outgoing messages as follows:

```
ex.setcounters(&txcounter,&rxcounter);
```

txcounter and rxcounter should be unsigned long integers. The class will automatically increment the value of these variables. The programmer must reset them to zero and display them on the screen as appropriate.

Additional Specialist functions

Only retained for backwards compatibility – do not use as described here.

The following functions are implemented to control the dynamics of the host infodriver's operation mode:

The mode constants are found in the <bncs.h> header file and are defined as follows:

```
#define IFMODE_NONE         0
#define IFMODE_RXONLY       1
#define IFMODE_TXRX         2
```

Set the mode of the infodriver as follows:

```
ex.setmode(iMode);
```

The return value is the new mode

Read the current mode of the infodriver as follows:

```
iMode=ex.getmode();
```

Control the dynamics of the automatic mode change facility as follows:

gpex->requestmode= ***OR-combination of request flags***

```
TO_TXRX         // this permits transitions to TX/RX mode
TO_RXONLY       // this permits transitions to RX only mode

example:        ex.requestmode=TO_TXRX | TO_RXONLY;
```

The default infodriver behaviour is the default value assigned to requestmode, i.e. **TO_TXRX**.
This means that the infodriver can auto-transition from a **RX Only** state to a **TX/RX** state, but not back again.
It is possible to change the auto-transition behaviour by changing the requestmode parameter. This can be done at any time.

Dynamic mode changes are notified to the callback function, with the iStatus member variable preset to either TO_TXRX or TO_RXONLY. However the behaviour is predetermined by the requestmode parameter.

Setmode(int) – updated November 2005 –not for CXinfo

Calling setmode(int iMode) will send messages to the infodriver (v2.1.1.4 or later) immediately, if appropriate, as well as setting the requestmode, you can set the requestmode directly if you don't want the action immediately.

Setting the modes does the following:-

- IFMODE_NONE

  - Not recommended!
  - No immediate action
  - Response to BBC_REQDEVGORXONLY is true
  - Response to BBC_INQDEVGOTXRX is false
- IFMODE_RXONLY
  - To be used when the driver wants to go RxOnly, regardless of if there is another driver available - not normal otherwise can not report comms fail except when it is a slave driver*.
  - Immediate action to go RXOnly
  - Response to BBC_REQDEVGORXONLY is true
  - Response to BBC_INQDEVGOTXRX is false
  - Note that setting both main and reserve to this will have no driver in TxRx
- IFMODE_RXONLYBROKEN
  - To be used when the driver will go RxOnly if another driver wants to go TxRx
  - No immediate action - Does not send an NB as the infodriver doesn't do this
  - Response to BBC_REQDEVGORXONLY is true
  - Response to BBC_INQDEVGOTXRX is false
  - Note that setting both main and reserve to this will have one driver in TxRx assuming you started with both in TxRx
- IFMODE_TXRX
  - To be used when the driver wants to go TxRx if another driver will allow it.
  - Immediate action is an NA on the network
  - Response to BBC_REQDEVGORXONLY is false
  - Response to BBC_INQDEVGOTXRX is true
- IFMODE_TXRXINQ
  - To be used when the driver can go TxRx if another driver will allow it - not normal operation
  - No immediate action
  - Response to BBC_REQDEVGORXONLY is false
  - Response to BBC_INQDEVGOTXRX is true
- IFMODE_FORCETXRX

- To be used when the driver must go TXRX, used if a slave driver*
- Send an NO on the network
- Response to BBC_REQDEVGORXONLY is false
- Response to BBC_INQDEVGOTXRX is true

*Slave driver is where a single device is represented by several infodrivers, on of which is master, the rest are slave, when the master goes TxRx the slaves should do the same, when the master goes Rx Only the slaves should do the same.

All changes of state are passed on to the external.

Setmode(int) – updated December 2005 –CXinfo only

The setmode function in CXinfo passes the value you set straight to the infodriver in the same way as the extinfo class does, but when the infodriver request information about the ability to go TxRx etc. this is passed straight to the notify function of the inherited class to be dealt with by the driver, this has been done to ensure backwards compatability.