MidiShare™

Release Notes

version 1.80

GRAME - Computer Music research Lab.
9, rue du Garet BP 1185
FR - 69202 LYON CEDEX 01
Ph: +33 (0)4 720 737 00
Fax: +33 (0)4 720 737 01

Email: midishare@grame.fr
http://www.grame.fr/MidiShare/

Summary

Introduction

About the release 1.80

MidiShare version 1.80 is an Open Source Release: its source code is publicly available under the GNU Library General Public License. The kernel architecture has been slightly revised in order to facilitates porting on new platforms. A GNU/Linux version has been designed, based on this architecture.

The main change consists in removing the IO drivers from the kernel itself and in providing mechanisms to plug these drivers dynamically as external ressources. Therefore, a new manager, the Drivers Manager, is part of the MidiShare kernel architecture: it is in charge of the drivers activation, it also routes the events to their final destination according to the drivers setup.

This document describes these changes and gives the reference of the new functions and data structures introduced with this new architecture. The reader is supposed to be familiar with the MidiShare Developer Documentation.

Overview of the MidiShare Drivers Management

The underlying idea of the drivers management is that a driver behaves like any MidiShare client except for a special call required to register (or unregister) as a driver. All the MidiShare API, extended by a set of new functions detailed below, are available to the drivers. Therefore, input / output operations may be performed using the standard MidiShare mecanisms:

- MidiShare events to be send to the physical ports may be handled in real-time using a MidiShare received alarm.
- events received by the driver from the physical ports may be given to the MidiShare kernel using standard MidiSend functions.

General operation

Loading drivers in memory is a platform dependant operation (so as unloading), in charge of the Drivers Manager. It is the driver responsability to register and unregister: it may be done at any time. However, the driver should act according to the MidiShare state:

- when MidiShare is dormant, the driver should be dormant
- when MidiShare wakes up, the driver should also be woken.

Therefore, in order to be notified of the kernel state changes, a driver should provide callbacks at register time. The TDriverOperation data structure is intended to store these callbacks:

```
typedef struct TDriverOperation {
   void     (* wakeup) (void);
   void     (* sleep)  (void);
   long     reserved[3];
} TDriverOperation;
```

The following functions are provided for drivers management :

```
short       MidiRegisterDriver  (TDriverInfos * infos, TDriverOperation *op)
void        MidiUnregisterDriver(short refnum)
short       MidiCountDrivers    ()
short       MidiGetIndDriver    (short index)
Boolean     MidiGetDriverInfos  (short refnum, TDriverInfos * infos)
```

The MidiShare driver

Like the MidiShare application, automatically opened with the first client, a MidiShare driver is created at wakeup time: it should be considered as the driver image of the MidiShare application. As MidiShare client applications should connect to MidiShare to communicate with the drivers, the MidiShare drivers should connect to the driver image of MidiShare to communicate with the client applications.

The MidiShare reference number is 0 for the client applications and 127 for the drivers.

Slots Management

To differenciate between MidiShare logical ports and input / ouput ports supported by a driver (Serial ports, USB...) we'll talk about 'slots' when we'll refer to a driver port.

A driver should declare all its available slots using the MidiAddSlot function, which returns a system unique slot reference number. This reference number is composed of the driver reference number and a slot reference number unique in the driver context.

```
  typedef struct {
     short  drvRef;   /* the driver reference number */
     short  slotRef;  /* the slot reference number   */
  } SlotRefNum;
```

The following functions are provided for the slots management :

```
SlotRefNum  MidiAddSlot     (short refnum, SlotName name,
                             SlotDirection direction)
SlotRefNum  MidiGetIndSlot  (short refnum, short index)
void        MidiRemoveSlot  (SlotRefNum slot)
Boolean     MidiGetSlotInfos(SlotRefNum slot, TSlotInfos * infos)
```

MidiShare allows dynamic routing of the events from one port to any of the available slots. For this reason, a slot can be connected to one or several MidiShare ports. The Driver Manager is in charge of dispatching the events to or from the drivers according to the connections set between the event port and the drivers slots.

The following functions are provided for the slot connections management :

```
void        MidiConnectSlot (short port, SlotRefNum slot, Boolean state)
Boolean     MidiIsSlotConnected (short port, SlotRefNum slot)
```

Additionnal change codes

The following alarm codes are defined to notify the client applications of changes concerning the drivers:

```
  MIDIOpenDriver      /* notified when a driver opens */
  MIDICloseDriver     /* notified when a driver closes */
  MIDIAddSlot         /* notified when a slot is added to the system */
  MIDIRemoveSlot      /* notified when a slot is removed from the system */
  MIDIChgSlotConnect  /* notified when the slots connections change */
```

Connections restrictions

Regular MidiShare clients and MidiShare drivers are considered to be part of separate communication folders. An additionnal connection rule is given : a connection is not allowed to cross a folder boundary. Therefore, connections can be made between regular clients or between drivers but NOT between a driver and a regular client. Such connections requests will be silently ignored by the kernel.

Obsolete functions

The functions below are obsolete:

```
Boolean     MidiGetPortState (short port)
void        MidiSetPortState (short port, Boolean state)
```

For compatibility, the corresponding entry points are supported in the Macintosh version but the functions do nothing and MidiGetPortState always return true

Reference

MidiAddSlot

Description

Add a new slot to a driver. MidiShare applications owning a "context alarm" will be informed of the new slot with the MIDIAddSlot alarm code.

Prototype

```
SlotRefNum MidiAddSlot (short drvRefNum, SlotName name,
                        SlotDirection direction);
```

Arguments

drvRefNum :	the MidiShare reference number of the owner driver.
name :	a slot symbolic name.
direction :	the slot direction which may be in `MidiInputSlot`, `MidiOutputSlot` or `MidiInputOutputSlot`

Result

The result is a 32-bit integer, a MidiShare unique slot reference number.

Note: *A SlotRefNum is composed of the driver reference number associated to a unique number in the driver context. Its structure is defined as:*

```
  typedef struct {
     short drvRef;
     short slotRef;
  } SlotRefNum;
```

A macro defined as:

```
  #define Slot(ref) (((ref).slotRef)
```

gives the slot reference number.

This part is the base of the Driver Manager events dispatching : events sent to MidiShare by a driver or given to a driver by MidiShare should always carry their slot reference number in their port field.

MidiConnectSlot

Description

Make or remove a connection between a slot and a MidiShare logical port. MidiShare applications owning a "context alarm" will be informed of the connection change using the MIDIChgSlotConnect alarm code.

Prototype

```
void MidiConnectSlot (short port, SlotRefNum slot, Boolean state);
```

Arguments

port :	a 16-bit integer, it is a MidiShare port number.
slot :	a 32-bit integer, it is the slot reference number.
state :	a Boolean value to specify whether the connection should be done (true) or removed (false).

Note: *a slot may be connected to one or several ports. The connection set determines the routing in the two directions:*

- *from MidiShare to the drivers: every slot connected to a port will get a copy of the events adressed to this port.*
- *from the drivers to MidiShare: every port connected to a slot will get a copy of the events received from this slot.*

MidiCountDrivers

Description

Gives the number of MidiShare drivers currently registered.

Prototype

```
short  MidiCountDrivers  ();
```

Result

The result is a 16-bit integer, the number of currently registered drivers

Example (ANSI C)

Print the name of all the registered MidiShare drivers

```
void PrintDriverNames(void)
{
  short  ref;
  short  i, n = MidiCountDrivers();

  printf( "List of MidiShare drivers :\n" );
  for( i = 1; i <= n; i++ )
  {
     ref = MidiGetIndDriver(i);
     printf("%i : %s \n", ref, MidiGetName( ref ) );
  }
}
```

Description

Gives information about a driver: it includes the driver name, its version number and its slots count.

Prototype

```
Boolean MidiGetDriverInfos (short ref, TDriverInfos * infos);
```

Arguments

`ref` : a 16-bit integer, it is the reference number of the driver.

`infos`: a pointer to a TDriverInfos structure to be filled with the driver characteristics.

Structure

description of the TDriverInfo structure :

```
typedef struct TDriverInfos {
     DriverName  name;
     short       version;
     short       slots;
     long        reserved[2]; /* reserved for future use */
} TDriverInfos;
```

the name field :	contains the driver name.
the version field :	contains the driver version number
the slots field :	contains the number of slots declared by the driver.

Result

The result is a Boolean value which indicates whether MidiShare has been able to get information about the driver or not.

Example (ANSI C)

Print information about all the registered MidiShare drivers

```
void PrintDriverInfos(void)
{
  short  ref; TDriverInfos infos;
  short  i, n = MidiCountDrivers();

  printf( "List of MidiShare drivers :\n" );
  for( i = 1; i <= n; i++ )
  {
     ref = MidiGetIndDriver(i);
     if (MidiGetDriverInfos (ref, &infos))
       printf("%i : %s v%d.%d - %d slots\n", ref, infos.name
                 infos.verson/100, infos.version%100, infos.slots );
  }
}
```

MidiGetIndDriver

Description

Gives the reference of number of a driver from its order number. The MidiGetIndDriver function allows to know the reference number of any driver by giving its order number (a number between 1 and MidiCountDrivers()).

Prototype

```
short MidiGetIndDriver (short index);
```

Arguments

`index`: a 16-bit integer, is the index number of a driver between 1 and MidiCountDrivers().

Result

The result is a driver reference number or MIDIerrIndex if the index is out of range.

Description

Gives the reference of number of a driver slot from its order number. The MidiGetIndSlot function allows to know the reference number of all the slots declared by a driver by giving its order number.

Prototype

```
SlotRefNum MidiGetIndSlot (short ref, short index);
```

Arguments

ref : a 16-bits integer, it is a driver reference number.

index : a 16-bits integer, it is the index number of the slot, between 1 and the TDriverInfos::slots field.

Result

The result is a 32-bit integer, the slot reference number,

or MIDIerrRefNum if the reference number is incorrect,

or MIDIerrIndex if the index is out of range.

MidiGetSlotInfos

Description

Gives information about a slot: it includes the slot name, the slot direction and its connection set to the MidiShare ports

Prototype

```
Boolean MidiGetSlotInfos (SlotRefNum ref, TSlotInfos * infos);
```

Arguments

ref : a 32-bit integer, the slot reference number.

infos: a pointer to a TSlotInfos structure to be filled with the slot characteristics.

Structure

description of the TSlotInfo structure :

```
typedef struct TSlotInfos {
     SlotName         name;
     SlotDirection    direction;
     char             cnx[32];
     long             reserved[2]; /* reserved for future use */
} TSlotInfos;
```

the name field : contains the slot name.
the direction field : indicates the slot direction, defined like below
the cnx field : a 256 bits field to indicates the 256 ports connections state.

```
typedef enum {     MIDIInputSlot=1,
                   MIDIOutputSlot,
                   MIDIInputOutputSlot } SlotDirection;
```

Result

The result is a Boolean value which indicates whether MidiShare has been able to get information about the slot or not.

Example (ANSI C)

Print information about a driver slots.

```
void PrintSlotsInfos(short driverRef)
{
  TDriverInfos dInfos; TSlotInfos sInfos;

  if (MidiGetDriverInfos (driverRef, &dInfos)) {
     short  i, j; SlotRefNum ref;
     printf( "List of %s slots :\n", dInfos.name);
     for( i = 1; i <= dInfos.slots; i++ )
     {
       ref = MidiGetIndSlot(driverRef, i);
       if (MidiGetSlotInfos (ref, &sInfos)) {
         printf("\n%i : %s : ", ref, sInfos.name);
         switch (sInfos.direction)
         {
           case MIDIInputSlot:
             printf("input slot");
             break;
           case MIDIOutputSlot:
             printf("output slot");
             break;
           case MIDIInputOutputSlot:
             printf("input/output slot");
             break;
         }
         printf(" : connected ports : ");
         for (j = 0; j < 256; j++) {
           if (IsAcceptedBit (dInfos.cnx, i))
             printf("%d ", i);
         }
       }
     }
  }
}
```

MidiIsSlotConnected

Description

Gives the state of a connection between a MidiShare port and a driver slot.

Prototype

```
Boolean MidiIsSlotConnected (short port, SlotRefNum ref);
```

Arguments

port :	a 16-bits integer, it is a MidiShare port number
ref :	the slot reference number

Result

The result is true when a connection exist between the port and the slot and false otherwise.

MidiRegisterDriver

Description

MidiRegisterDriver is similar to MidiOpen except that the application is registered as a driver instead of a regular MidiShare client. At register time, a driver should give the necessary callbacks to be notified of the MidiShare wakeup and sleep transitions. Any registered driver must call the MidiUnregisterDriver function before leaving, by giving its reference number as an argument. MidiShare can thus be aware of the precise number of active Midi drivers. MidiShare applications owning a "context alarm" will be informed of the new driver using the MIDIOpenDriver alarm code

Prototype

```
short MidiRegisterDriver (TDriverInfos * inf, TDriverOperation *op);
```

Arguments

inf : a pointer to a structure containing drivers information.

op : a pointer to a structure containing the callbacks required to be notified of the MidiShare state changes.

Structure

description of the TDriverInfo structure :

```
  typedef struct TDriverInfos {
     DriverName  name;           /* the driver name */
     short       version;        /* its version number */
     short       slots;          /* ignored at register time */
     long        reserved[2];
  } TDriverInfos;
```

description of the TDriverOperation structure :

```
  typedef struct TDriverOperation {
     void     (* wakeup) (short refNum);
     void     (* sleep)  (short refNum);
     long     reserved[3];
  } TDriverOperation;
```

the wakeup field :	points to a function called by MidiShare at wakeup time.
the sleep field :	points to a function called by MidiShare at sleep time.
the reserved fields :	reserved for future use. They should be set to zero.

Result

The result is a unique reference number identifying the driver.

Note: *when registering, a driver should take care of the following: if MidiShare is running at register time, the driver 'wakeup' callback will be called before the MidiRegisterDriver function return. Therefore, a correct 'wakeup' callback implementation should not rely on code executed after the MidiRegisterDriver function.*

MidiRemoveSlot

Description

Remove a slot from the system. MidiShare applications owning a "context alarm" will be informed of the change using the MIDIRemoveSlot alarm code.

Prototype

```
void MidiRemoveSlot (SlotRefNum ref);
```

Arguments

ref : a slot reference number.

MidiUnregisterDriver

Description

This is used for closing of a MidiShare driver. MidiShare applications owning a "context alarm" will be informed of this closing using the MIDICloseDriver alarm code.

Prototype

```
void MidiUnregisterDriver (short ref);
```

Arguments

ref : a driver reference number, given by the corresponding MidiRegisterDriver call.

Note *: when unregistering, a driver should take care of the following: if MidiShare is active at unregister time, it will first put the driver to sleep by calling the driver 'sleep' callback.*